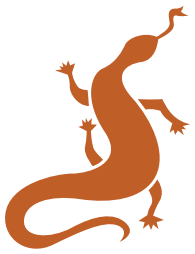


Goanna Central User Guide

Version 3.6.4
Linux/Windows Edition

November 24, 2015

© 2008-2015, Red Lizard Software

Copyright © 2008-2015 Red Lizard Software
All rights reserved.

This document, as well as the software described in it, is provided under license and may only be used or copied in accordance with the terms of such license. The information contained herein is the property of NICTA and is made available under license to Red Lizard Software. It is confidential information as between the Recipient and Red Lizard Software and remains the exclusive property of NICTA. No part of this documentation may be copied, translated, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of NICTA.
NICTA does not warrant that this document is error-free.

Red Lizard Software
Australian Technology Park
Level 5, 13 Garden Street
Eveleigh NSW 2015
Australia

Web: http://www.redlizards.com
Support: support@redlizards.com

Contents

1 System Requirements **5**

1.1 Operating Systems . . . 5

1.1.1 Microsoft Windows . . . 5

1.1.2 Linux . . . 5

1.1.3 Other Requirements . . . 5

1.2 Hardware Requirements . . . 7

1.3 Supported Compilers . . . 7

1.3.1 A Word On C99 and C++11 Support . . . 7

1.3.2 A Word On Compiler-Specific Syntax Extensions . . . 8

1.3.3 Analog Devices C/C++ Compiler for SHARC Processors (`cc21k` Dialect) . . . 8

1.3.4 ARM C/C++ Compiler (`armcc` Dialect) . . . 8

1.3.5 Cosmic Software C Cross Compiler (`cosmic` Dialect) . . . 8

1.3.6 Freescale (`metrowerks` Dialect) . . . 8

1.3.7 GNU C/C++ Compiler (GCC) (gnu Dialect) . . . 8

1.3.8 IAR Toolchain for 8051, ARM, AVR, MSP430, M16C and R8C (`iar-8051`, `iar-arm`, `iar-avr`, `iar-msp430` and `iar-m16c` Dialects) . . . 9

1.3.9 Keil Cx51 and C166 Optimizing C Compiler (`c51` and `c166` Dialects) . . . 9

1.3.10 Microsoft Visual C++ (`microsoft` Dialect) . . . 9

1.3.11 QNX QCC (`qnx` Dialect) . . . 9

1.3.12 Renesas H8S, H8/300 Series C/C++ Compiler (`renesas-h8` Dialect) . . . 9

1.3.13 Renesas RXC Toolchain (`renesas-rx` Dialect) . . . 9

1.3.14 TI Build Tools (`ti-armcl`, `ti-cl16x`, `ti-cl2000`, `ti-cl430`, `ti-cl470`, `ti-cl500` and `ti-cl55` Dialects) . . . 10

1.3.15 Wind River Diab Compiler (`diab` Dialect) . . . 10

1.4 Supported Build Systems . . . 10

2 Getting Started **12**

2.1 License Agreement . . . 12

2.2 Installation (Linux) . . . 12

2.3 Installation (Windows) . . . 13

2.4 License Activation . . . 13

2.4.1 Activating Node-locked License . . . 13

2.4.2 Using Network (Floating) License . . . 14

2.5 Next Steps . . . 14

3 Setting Up Projects for Goanna Analysis **16**

3.1 Introduction . . . 16

3.2 Initialising the Goanna directory . . . 16

3.2.1 Using a different Goanna directory . . . 17

3.2.2 Using a different project directory . . . 17

3.3 Performing a build trace . . . 18

4 Running Goanna Analysis **19**
4.1 Introduction . . . 19
4.2 Running Goanna project analysis using `goanna-analyse` . . . 19
4.2.1 Running Goanna project analysis on a subset of the project . . . 19
4.3 Reading Analysis Results . . . 20
4.3.1 Goanna Output On The Console . . . 21
4.3.2 HTML Report of Analysis Results . . . 22
4.3.3 Analysis Results In XML File . . . 23
4.3.4 Using Goanna Dashboard Web Interface To Interact With Analysis Results . . . 23
4.4 Running Goanna Analysis using pre-3.4.0 tools . . . 23

5 Configuring Goanna Analysis **24**
5.1 Checks Packages . . . 24
5.1.1 Listing Checks Packages . . . 24
5.1.2 Enabling Available Checks Package . . . 24
5.1.3 Disabling Installed Checks Package . . . 25
5.1.4 Installing Custom Checks Package . . . 25
5.2 Setting Checks . . . 25
5.2.1 Introduction . . . 25
5.2.2 Selecting Checks With Command Line Options . . . 25
5.2.3 Enabling All Available Checks . . . 26
5.2.4 Selecting Checks for a C/C++ Project . . . 26
5.3 Including Headers Into Analysis . . . 27
5.4 Setting Analysis Timeouts . . . 27
5.5 Ignoring Certain Warnings ("Warning Suppression") . . . 27
5.6 Using Knowledge Base To Guide Analysis . . . 27
5.7 Other Configuration Options . . . 28

6 Running Goanna Analysis From Within IDEs **29**
6.1 Running Goanna Analysis From IAR Embedded Workbench® . . . 29
6.2 Running Goanna Analysis From Keil™ μVision® . . . 30

7 Getting the Best Results from Goanna **33**
7.1 Interprocedural Analysis . . . 33
7.2 A Word on False Positives . . . 33
7.3 Using the `_GOANNA` Preprocessor Symbol . . . 34
7.4 Using the `assert` macro . . . 34
7.5 Sample Code . . . 34

8 Using Goanna To Compute Code Metrics **35**
8.1 Introduction . . . 35
8.2 Using `goanna-metrics` To Compute Metrics . . . 35
8.2.1 Computing Code Metrics For Subset Of Project . . . 36
8.2.2 Code Metrics Results In XML, HTML and CSV Files . . . 36

9 Using the Goanna Dashboard **37**
9.1 Getting to the Goanna Dashboard . . . 37
9.2 Bug Statuses . . . 37
9.3 Severity . . . 37
9.4 Dashboard Views . . . 37
9.4.1 Project Page . . . 37
9.4.2 Report Page . . . 38
9.4.3 Directory Browser . . . 39
9.4.4 Warnings Browser . . . 40
9.4.5 Code Browser . . . 41
9.5 Database Upgrades . . . 42
9.6 Project Settings (Advanced) . . . 43
9.6.1 Code Browser Character Encodings . . . 43

10 Advanced Features, Concepts and Configurations **44**
10.1 Proceed With Caution . . . 44
10.2 Manually Running Analysis On Source Files . . . 44
10.3 Manually Running Link Time Analysis On Object Files . . . 45
10.4 The Project Database . . . 46
10.5 How Goanna's Compiler Support Work . . . 47

11 Goanna Central Utility Reference **48**
11.1 `goanna-init` – Initialise C/C++ Projects . . . 48
11.2 `goanna-trace` – Monitor C/C++ Project Build . . . 49
11.3 `goanna-analyse` – Analyse C/C++ Projects . . . 50
11.4 `goanna-report` – Generate C/C++ Project Reports . . . 53
11.5 `goanna-snapshot` – Upload C/C++ Project Snapshots . . . 54
11.6 `goanna-package` – Customise C/C++ Project Analysis . . . 55
11.7 `goanna-metrics` – Calculate C/C++ Project Code Metrics . . . 56
11.8 `goanna-assume` – Import/Export Knowledge Base . . . 58
11.9 `goannacc` and `goannac++` – Analyze C/C++ Source Files . . . 64
11.10 `goannald` – C/C++ Link Time Analysis . . . 69
11.11 `goreporter` – Goanna Dashboard Server and Administration Tool . . . 71

Index **75**

1 System Requirements

Before using Goanna, please check that your system and project meets the system requirements.

1.1 Operating Systems

1.1.1 Microsoft Windows

Goanna supports the following versions of Windows:

- Windows XP (Service Pack 2 or higher)
- Windows Vista
- Windows 7
- Windows 8
- Windows 8.1
- Windows 10
- Windows Server 2003 (Service Pack 1 or higher)
- Windows Server 2008
- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2012 R2

Both 32-bit (x86) and 64-bit (x86-64/AMD64) versions of Windows are supported (except Windows XP and Windows Server 2003, which we only support 32-bit version).

Required Software

Before installing Goanna, you will need to install the following:

- Microsoft Visual C++ 2008 Redistributable (Download from http://www.microsoft.com/en-us/download/details.aspx?id=5582)
- .NET Framework 2.0 or higher

1.1.2 Linux

Goanna supports all major distributions of Linux with glibc (GNU C Library) 2.4 or higher installed.

Both 32-bit (x86) and 64-bit (x86-64/AMD64) versions of Linux are supported.

Using Goanna with SELinux enabled is not recommended.

1.1.3 Other Requirements

Some features of Goanna may require additional software or packages.

Requirements for Goanna Dashboard and HTML Report

Goanna Dashboard (see 9) and HTML Report (see 4.3.2) requires a web browser. The following web browsers are supported:

- Internet Explorer 9 or higher
- Microsoft Edge
- Mozilla Firefox - currently supported versions by Mozilla
- Google Chrome - currently supported versions by Google

While not officially supported, these features may also work on other modern browsers.

We also support Internet Explorer 7 and 8, however you may experience slow performance on these browsers; using Goanna Central with these browsers is not recommended.

Requirements for LM-X License Manager

Goanna uses LM-X License Manager 4.4.2 from X-Formation for licensing.

If you wish to use web-based UI of the License Manager (**Important:** Red Lizard Software does not provide full support for the web-based UI), the following software must be installed:

- A modern web browser
- Oracle Java Runtime Environment 1.6 or higher
- Adobe Flash Player

Please refer to X-Formation website (http://docs.x-formation.com/display/GEN/System+requirements+for+web-based+UIs) for more information.

Requirements for SonarQube Integration

Important: Red Lizard Software only provides support for Goanna SonarQube plugin, not for SonarQube itself. If you require assistance in setting up, or using SonarQube dashboard, please refer to SonarQube website at http://www.sonarqube.org.

Red Lizard Software provides a separate SonarQube plugin to publish the analysis result to SonarQube dashboard (previously called "Sonar"). You can download this plugin and the user manual from http://www.redlizards.com customer portal. (You will need customer user account to download)

Please refer to the SonarQube plugin manual for requirements to use this plugin. For requirements for SonarQube, please refer to SonarQube website at http://www.sonarqube.org.

Notes

- SonarQube integration for projects using deprecated `goanna` or `gokeil` commands is no longer available as of Goanna Central 3.5.0.
- SonarQube integration for SonarQube 4.2 and older is no longer available as of Goanna Central 3.5.0. If you are using these old versions of SonarQube, you will need to upgrade your installation of SonarQube.
- Goanna Central no longer bundles Goanna customised version of SonarQube as of Goanna Central 3.5.0. If you are using this version of SonarQube, you will need to upgrade to the official distribution of SonarQube 4.3 or higher.

1.2 Hardware Requirements

Goanna requires, at the minimum, the following hardware:

- Processor: Intel Pentium 4 or higher
- Memory: 1 GB or more
- Storage: Minimum 1 GB of free disk space

For optimal analysis performance, we recommend at least the following:

- Processor: Intel Core 2 Duo or later CPU with minimum speed 2 GHz. Multi core CPUs are recommended.
- Memory: 4 GB or more
- Storage: 5 GB or more of free disk space

For large projects, Goanna may require more RAM and disk space than the ones shown here.

1.3 Supported Compilers

Goanna currently supports the following compilers:

Compiler Name	Goanna Dialect Name	Common Compiler Executables
Analog Devices C/C++ Compiler for SHARC Processors	`cc21k`	cc21k
ARM C/C++ Compiler	`armcc`	armcc, armlink
Cosmic Software C Cross Compiler	`cosmic`	cx6808, cx6812, cx6816, cxs12x, cxstm8
Freescale (formerly Metrowerks)	`metrowerks`	mwccarm, mwccmcf
GNU	`gnu`	gcc, g++, ld
IAR Toolchain for 8051	`iar-8051`	icc8051
IAR Toolchain for ARM	`iar-arm`	iccarm
IAR Toolchain for AVR	`iar-avr`	iccavr
IAR Toolchain for MSP430	`iar-msp430`	icc430
IAR Toolchain for M16C and R8C	`iar-m16c`	iccm16c
Keil Cx51 Optimizing C Compiler	`c51`	c51, cx51, bl51, lx51
Keil C166 Optimizing C Compiler	`c166`	c166, l166
Microsoft Visual C++	`microsoft`	cl, link
QNX QCC	`qnx`	qcc
Renesas H8S, H8/300 Series C/C++ Compiler	`renesas-h8`	ch38
Renesas RXC Toolchain	`renesas-rx`	ccrx
TI Build Tools - armcl	`ti-armcl`	armcl
TI Build Tools - CL16X	`ti-cl16x`	cl6x
TI Build Tools - CL2000	`ti-cl2000`	cl2000
TI Build Tools - MSP430	`ti-cl430`	cl430
TI Build Tools - CL470	`ti-cl470`	cl470
TI Build Tools - CL500	`ti-cl500`	cl500
TI Build Tools - CL55	`ti-cl55`	cl55
Wind River Diab Compiler	`diab`	dcc

Notes

- Green Hills compiler is no longer supported as of Goanna 3.1.0.
- As of Goanna 3.4.0, support for Tasking C166 compiler is only available upon request. Contact support@redlizards.com if you require support for this compiler.
- Goanna also ships with `cygwin`, `gnu-4.4.4` and `ti` dialects. However, these dialects exist only for backward compatibility. We strongly recommend that you do not use these dialects.

1.3.1 A Word On C99 and C++11 Support

Goanna strives to support most C99 and C++11 features as long as the compilers used in your projects also accept them. However, please note that, for C++11 extensions, the analysis engine generally does not make use of these extensions or any information derived from usage of these extensions. For example, Goanna does not perform any pointer or memory use related analysis on C++11 `std::shared_ptr`.

1.3.2 A Word On Compiler-Specific Syntax Extensions

Goanna strives to support most compiler-specific C/C++ syntax extensions for supported compilers. However, please note that even in cases where Goanna supports compiler-specific C/C++ syntax extensions, the analysis engine will generally not make use of these extensions or any information derived from usage of these extensions.

For example, Goanna does not take Keil Cx51 Memory Models or Memory Types into account during analysis, even though Goanna supports relevant syntax extensions.

1.3.3 Analog Devices C/C++ Compiler for SHARC Processors (`cc21k` Dialect)

Goanna supports Analog Devices C/C++ Compiler for SHARC Processors shipped with Analog Devices CrossCore Embedded Studio 1.0 or higher.

1.3.4 ARM C/C++ Compiler (`armcc` Dialect)

Goanna supports the following versions of ARM C/C++ Compiler:

- RealView Development Suite (RVDS) versions 2.0 to 4.1 (inclusive)
- DS-5 Development Studio
- ARM Compiler versions 4.1 and 5.0 (including those shipped with Keil MDK-ARM versions 4 and 5)

RVDS 1.2 and older, ARM Developer Suite (ADS) versions 1.2 and older, and ARM Compiler 6 (`armclang`) are not supported.

Known Limitations

- Some compiler arguments, such as `--cpp` (for `--cpp` use `--c++` instead), `--kandr_include`, `--strict`, `--wchar`, `--no_wchar`, `--wchar16` and `--wchar32` are not supported. Goanna will ignore these arguments.
- Goanna will always include `RVCT<version>INC`, `ARMCC<version>INC` and `ARMINC` environment variables, and `<installation-path-of-compiler>\..\include` into the system include directories, even when `-J` option is used.

1.3.5 Cosmic Software C Cross Compiler (`cosmic` Dialect)

Goanna supports all recent versions of:

- CX6808 Compiler (part of Cosmic S08 and HC08 Cross Development Tools)
- CX6812 Compiler (part of Cosmic 68HC12 and HCS12 Cross Development Tools)
- CX6816 Compiler (part of Cosmic 68HC16 Cross Development Tools)
- CXS12X Compiler (part of Cosmic S12X and XGATE Cross Development Tools)
- CXSTM8 Compiler (part of Cosmic STM8 Cross Development Tools)

Note: CXXGATE Compiler (part of Cosmic S12X and XGATE Cross Development Tools) is not supported.

1.3.6 Freescale (`metrowerks` Dialect)

Goanna supports `mwccarm` compiler shipped with Freescale CodeWarrior Development Studio for Microcontrollers (CW MCU) version 10.2 or higher.

1.3.7 GNU C/C++ Compiler (GCC) (gnu Dialect)

Goanna supports most versions of GNU C/C++ Compiler (GCC).

Known Limitations

- `CPATH`, `C_INCLUDE_PATH` and `CPLUS_INCLUDE_PATH` environment variables are not recognized by Goanna.
- `--sysroot` option is not supported.
- Goanna ignores all optimisation options to work around known issues in recognising certain macros in GNU libc. Goanna may produce unexpected results if your source code relies on `__OPTIMIZE__`, `__OPTIMIZE_SIZE__`, or `__NO_INLINE__` predefined macros.

1.3.8 IAR Toolchain for 8051, ARM, AVR, MSP430, M16C and R8C (`iar-8051`, `iar-arm`, `iar-avr`, `iar-msp430` and `iar-m16c` Dialects)

Goanna supports IAR Toolchain for 8051, ARM, AVR, MSP430, M16C and R8C shipped with IAR Embedded Workbench 5.40 or higher.

Note: IAR Toolchain for AVR32 is not supported.

Known Limitations for IAR Toolchain for AVR compiler (`iar-avr` dialect)

- Goanna does not recognize `QCCAVR` environment variable.
- Goanna always recognize IAR syntax extensions. `-e` and `--strict` options, and `#pragma language` directive are ignored.
- C++ support for this compiler is limited. If your source code uses some standard library headers, such as `<vector>`, parse errors may occur. Additionally, `--ec++` or `--eec++` option causes Goanna to recognize full C++ language features, rather than Embedded C++ subset.

1.3.9 Keil Cx51 and C166 Optimizing C Compiler (`c51` and `c166` Dialects)

Goanna supports all recent versions of Keil Cx51 and C166 Optimizing C Compiler.

1.3.10 Microsoft Visual C++ (`microsoft` Dialect)

Goanna supports Microsoft Visual C++ compiler shipped with Microsoft Visual Studio 6.0 or higher.

Known Limitations

- Managed C++, C++/CLI and C++/CX syntax extensions, and CLR (Common Language Runtime) related compiler options are not supported.

1.3.11 QNX QCC (`qnx` Dialect)

Goanna supports QNX QCC shipped with QNX Momentics 4.7.0 or higher.

1.3.12 Renesas H8S, H8/300 Series C/C++ Compiler (`renesas-h8` Dialect)

Goanna supports `ch38` compiler shipped with Renesas C/C++ Compiler Package for H8SX, H8S, H8 Family (also called "H8S, H8/300 Series C/C++ Compiler") version 7.00 or higher.

1.3.13 Renesas RXC Toolchain (`renesas-rx` Dialect)

Goanna supports CC-RX compiler shipped with Renesas C/C++ Compiler Package for RX Family version 1.02 or higher.

1.3.14 TI Build Tools (`ti-armcl`, `ti-cl16x`, `ti-cl2000`, `ti-cl430`, `ti-cl470`, `ti-cl500` and `ti-cl55` Dialects)

Goanna supports the following Texas Instruments compilers:

- armcl compiler shipped with Texas Instruments Code Composer Studio versions 5 and 6,
- CL16X compiler shipped with Texas Instruments Code Composer Studio versions 2.2, 3 and 5,
  Note: CL16X compiler shipped with Texas Instruments Code Composer Studio version 4 is not supported.
- CL2000 compiler shipped with Texas Instruments Code Composer Studio versions 4 and 5,
- CL430 compiler shipped with Texas Instruments Code Composer Studio versions 4 and 5,
- CL470 compiler shipped with Texas Instruments Code Composer Studio version 5,
- CL500 compiler shipped with Texas Instruments Code Composer Studio version 5, and
- CL55 compiler shipped with Texas Instruments Code Composer Studio version 5.

Note: Except for armcl, compilers shipped with Texas Instruments Code Composer Studio version 6 are not supported.

Known Limitations

- Goanna recognizes compiler intrinsics for CL2000 compiler; however, they are not used for analysis.
- Goanna recognizes limited number of compiler intrinsics for armcl compiler; however, they are not used for analysis.
- Goanna does not recognize compiler intrinsics for any other Texas Instruments compiler; parse errors may occur if your source code uses them.
- Goanna does not recognize environment variables used to specify command line arguments.
- Except for armcl, Goanna does not recognize environment variables used to specify include paths.

1.3.15 Wind River Diab Compiler (`diab` Dialect)

Goanna supports Wind River Diab Compiler shipped with Wind River Workbench 3.3 or higher.

1.4 Supported Build Systems

There is no specific requirement on a build system you can use (except for certain build systems listed below). Goanna Central comes with a generic build integration system that allows Goanna to integrate with most build systems.

In addition, Goanna Central provides a mechanism to run analysis from the following IDEs:

- IAR Embedded Workbench 5.40 or higher (see 6.1 for instructions)
- Keil μVision 4 or higher (see 6.2 for instructions)

Build Systems with Special Requirements

Goanna Central also supports the following build systems, but the special requirements apply.

Cygwin (Windows only)

- Goanna requires Cygwin 1.7.23 or higher.
- Goanna requires `cygpath.exe` to be present inside `/usr/bin` directory. The official Cygwin installer should have installed this binary by default. Parse errors may occur if this binary is not present.
- All file and directory path options to Goanna utitilies must be specified in Windows format. Goanna utilities do not accept Cygwin style paths.
- `wincmdln` option in `CYGWIN` environment variable must be set prior to using Goanna to ensure Goanna can capture build information correctly.

MSYS2 (Windows only)

- Goanna does not support compilers targetting the MSYS2 environment (such as in `gcc` package). Note that standard MinGW compilers (such as in `mingw-w64-i686-gcc` package) are supported (even in `MSYS2` environment).

MSYS (Windows only)

- Goanna does not support compilers targetting the MSYS environment (shipped as part of `msys-dvlpr` package). Note that standard MinGW compilers are supported (even in `MSYS` environment).

Other Known Limitations and Notes

- As of Goanna Central 3.4.0, support for Keil μVision is now available by default. The Keil μVision Support Package is no longer required, but is still available upon request for backward compatibility.

2 Getting Started

2.1 License Agreement

Before installing Goanna Central, ensure you read the Goanna license agreement.
For evaluation (trial) licenses, please refer to:

http://redlizards.com/license-term/evaluation-license-agreement/

For registered (paid) licenses, please refer to:

http://redlizards.com/license-term/

2.2 Installation (Linux)

To install Goanna Central for Linux:

1. Download the Goanna Central for Linux tarball.
2. Unpack the downloaded tarball:

   ```
   tar -zxvf goanna-central-linux-release-3.6.4.tgz
   ```

   This should extract all files needed for installation in a separate directory.
3. Navigate to the directory just created:

   ```
   cd goanna-central-linux-release-3.6.4
   ```

4. Run the install script (`install-goanna`) to start the installation process.
   The installation process can be run either with `sudo` (i.e. under root):

   ```
   sudo ./install-goanna
   ```

   or without `sudo` (i.e. under your user account):

   ```
   ./install-goanna
   ```

   We recommend that you install Goanna with `sudo`. The following features will not be installed if you do not install Goanna with `sudo`:
   - The Goanna Dashboard daemon; if the daemon is not installed, then to access the Goanna Dashboard, you will need to start the server manually. See 9 for details.
5. Follow the instructions of the install script to complete installation.
   By default, Goanna Central will be installed to:
   - `/usr/local/goanna` if installed with `sudo`, or
   - `$HOME/goanna` if installed without `sudo`.

   If you wish to install Goanna Central to other location, provide a desired location when the install script asks for the installation path.
6. To use Goanna Central from the command line, you should set your `PATH` environment variable to include Goanna's `bin` directory. The install script will show you how to do this at the end of installation.
   For example, if you installed Goanna in `/usr/local/goanna`, you can add:

   ```
   export PATH=$PATH:/usr/local/goanna/bin
   ```

   to your `~/.profile` file.

2.3 Installation (Windows)

To install Goanna Central for Windows:

1. Download the Goanna Central for Windows installer.
2. Double click the installer `.msi` file.
3. Follow the instructions of the installer to complete installation.

2.4 License Activation

Whether you are just evaluating Goanna or have purchased the full version, you must activate your license before you can use Goanna.

2.4.1 Activating Node-locked License

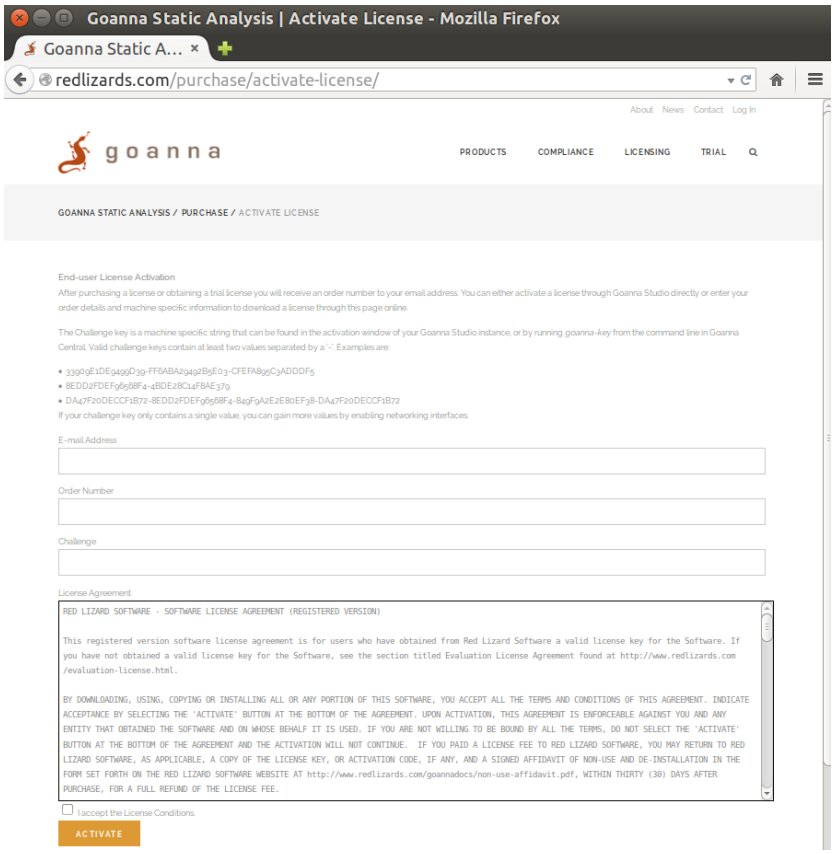
If you have obtained node-locked licenses, you should have received an email containing your license information from Red Lizard Software. This email will contain an Order Number which is required to complete the activation process.

Before activating your node-locked license, you will need a *challenge key* for your computer. To obtain the challenge key, run:

```
goanna-key
```

Now, to activate your node-locked license, follow these steps:

1. Go to the activation page at: http://www.redlizards.com/purchase/activate-license/



Enter the following details:

Email Address: The email address you provided when purchasing Goanna.

Order Number: The order number provided in the Goanna purchase confirmation email.

Challenge: The challenge key of the computer you wish to activate the license for.

2. Read the license agreement.
3. Check "I accept the License Conditions", and click "Activate" to accept the license agreement and activate your license.
4. The resulting license file (called `goanna_license.lic`) will be sent to the email address you provided. You will also be taken to a page where you can download the license file by clicking on Download your license.
5. Save the `goanna_license.lic` file to the following location:

   Linux `$HOME` (Home directory of the user running Goanna), or `/etc/goanna`

   Windows (XP and Server 2003) `C:\Documents and Settings\<username>\Local Settings\Application Data\RedLizards\Goanna Central`

   Windows (Vista, Server 2008 and later) `C:\Users\<username>\AppData\Local\RedLizards\Goanna Central`
   (where `<username>` is the name of the user account who have installed Goanna Central; usually it is your Windows user name).

2.4.2 Using Network (Floating) License

If you have obtained network (floating) licenses, then you will need to perform the following steps to set up Goanna to use your license:

1. Set up a license server somewhere on your network with the license file supplied.
   The license server for Goanna is shipped in a separate installation package. Contact your distributor to obtain the installer.
   For instructions on how to set up the license server, please refer to the separate "License Server User Guide".
2. Always use **`--license-server`** option when using Goanna.

Once the license server is up and running, to use the license server, pass the **`--license-server=<server>`** option to `goanna-trace`, `goanna-analyse`, `goanna-metrics` and `goannacc` utilities when capturing build information, running Goanna analysis, and computing code metrics.

Borrowing a Network (Floating) license

When you run the Goanna analysis with a network (floating) license, you can optionally specify a borrow duration (in the range of 1 to 24 hours). This is enabled by using the **`--license-borrow-hours=<hours>`** option in combination with the `--license-server` option, and will force Goanna to reserve a single license seat for you for this amount of time. In addition this technique can be used to improve license validation time, and will allow you to perform Goanna analysis while disconnected from the license server.

2.5 Next Steps

Performing analysis on your C/C++ project with Goanna Central is a three-step process:

1. Setting up your C/C++ project, and running a full build on the project using Goanna build integration utility to capture settings of your build.
2. Using this information from full build to run analysis.
3. Reading and interacting with the analysis results.

The rest of this documentation explains this process in detail:

1. The next section, section 3, explains how to set up your C/C++ projects to be used with Goanna, and to capture settings of your build.
2. The section 4 explains how to use this captured settings to run analysis, and to read the analysis results.

3. The section 5 explains what options are available to control the analysis, and how to do so.
4. The section 6 explains ways to set up IAR Embedded Workbench or Keil μVision IDEs to run analysis directly from within IDE.
5. The section 8 explains how to compute code metrics of your project.
6. The section 9 explains how to use the Goanna Dashboard, a web-based interface to navigate and interact with analysis results and issues found within your projects.

3 Setting Up Projects for Goanna Analysis

3.1 Introduction

The most basic way to perform Goanna analysis on the source code is *project-wide analysis* [1]. Project-wide analysis scans and identifies potential issues within the whole project, based on information about how the project is built.

To perform project-wide analysis, the following steps should be taken:

1. First, you need to ensure that the project compiles successfully with no syntax errors. Goanna analysis engine relies on the source code being syntactically correct.
2. Clean your projects source tree. This will ensure Goanna detects all of the project source files during build tracing.
3. Initialise a Goanna directory. This directory is used by Goanna to store information about your project.
4. Run a Goanna build tracing utility. This utility runs and monitors a full build of the project, and captures all necessary information for Goanna to understand the source files. This process is called *build tracing*, and the result of this process is stored in a file called *build specification*.
5. Run the project analysis utility with the generated build specification to perform project analysis.

The generated build specification file can be re-used for future Goanna analysis, so that you do not have to re-run full build before every analysis. However, if the structure or settings of the project changes, such as:

- When a new source file has been added to the project, or
- When an existing source file has been removed from the project, or
- When the project build settings themselves change, such as adding new include paths or predefined macros, or upgrading to a newer version of compiler,

then the full build, and build recording process should be performed again to ensure the build specification reflects the changes made to the project.

Use The Same Machine For Build Tracing And Project Analysis

To ensure that Goanna has an accurate understanding of the source files, Goanna needs access to not only the source files, but other extra build-specific information such as:

- All used header files
- Configuration of the compiler used

This means that Goanna requires that the machine used to analyze the project also has:

- The same version of the compiler used,
- The same versions of all used libraries, and
- The same version of the build program used.

It is therefore recommended to run analysis on the same machine used to build the project.

3.2 Initialising the Goanna directory

Before performing a build trace, it is necessary to initialise a Goanna project information storage directory (Goanna directory).

In the *project root directory*, run the following command:

```
goanna-init
```

[1] Goanna also provides other methods to perform analysis. See 10 for details.

```
Goanna Directory is: <current-working-directory>/.goanna
Project Directory is: <current-working-directory>
Detected Project name: <current-working-directory>
Created new Goanna Directory at: <current-working-directory>/.goanna
```

This will create a Goanna directory where the information Goanna gathers about your project will be stored. All Goanna directory based commands will then search for, write to, and use the information in the Goanna directory where applicable.

By default, `goanna-init` determines the name of the project from the name of the project root directory. This name is used for reporting purposes, such as HTML analysis report. If you wish to change the name of the project, pass `--project-name` option to `goanna-init`.

3.2.1 Using a different Goanna directory

If the default Goanna directory location is undesirable, then it can be created in a non-default location using the **`--goanna-dir`** command line option.

`goanna-init --goanna-dir=<other-dir>`

```
Goanna Directory is: <other-dir>/.goanna

Project Directory is: <current-working-directory>

Created new Goanna Directory at: <other-dir>/.goanna
```

In order to use this non-default Goanna directory from other Goanna directory based commands, it is necessary to also give them the **`--goanna-dir`** command line option.

3.2.2 Using a different project directory

If the current working directory is not also the *project root directory*, then this can also be specified using the **`--project-dir`** command line option.

`goanna-init --goanna-dir=<other-dir> --project-dir=<my-project-dir>`

```
Goanna Directory is: <other-dir>/.goanna

Project Directory is: <my-project-dir>

Created new Goanna Directory at: <other-dir>/.goanna
```

The **`--project-dir`** option cannot be given without also using the **`--goanna-dir`** option. If the non-default project directory is used, then both options also need to be given to Goanna directory based commands.

3.3 Performing a build trace

The command **goanna-trace** is used to monitor a build, and produce a build specification inside the Goanna directory.

Using goanna-trace To Capture The Build Settings

After the project has been cleaned, the command to run the build and capture the build settings is:

```
goanna-trace -- <build-command>
```

This will execute *<build-command>* and monitor the processes that are created.

For example, If the project is a GNU Makefile project, and it contains the following content:

```
example.o: example.c
  gcc -c -o $@ $<

example: example.o
  ld -o $@ $<
```

Then executing the command:

```
goanna-trace -- make example
```

will execute the following build steps,

```
gcc -c -o example.o example.c

ld -o example example.o
```

which will be captured in the Goanna build specification.

Notes

- If your build system is a script (e.g. shell script, Python script), you may need to specify the shell or the interpreter in the build command (e.g. **goanna-trace -- bash <path-to-script>** instead of **goanna-trace -- <path-to-script>**).
- On Linux, `goanna-trace` uses `LD_LIBRARY_PATH` and `LD_PRELOAD` environment variables to allow Goanna to monitor the build. If your build currently uses these environment variables, you need to modify the build to not use these environment variables before running `goanna-trace`.
- If you are using Cygwin, you must set `wincmdln` option in the `CYGWIN` environment variable before running `goanna-trace`.

You can now use `goanna-analyse` to perform a project analysis. See 4.2 for more details.

4 Running Goanna Analysis

4.1 Introduction

To perform a project analysis, make sure you have completed the steps required to generate a Goanna build specification that can be used by `goanna-analyse`.

4.2 Running Goanna project analysis using `goanna-analyse`

Run:

```
goanna-analyse
```

to start the whole project analysis.

It is important to note that the analysis may take a long time; if the project contains a large number of files (generally 100 or more), and/or large files, then the analysis may possibly take hours to complete.

To pass additional Goanna analysis options, add them to the end of the command. For example:

```
goanna-analyse <additional-analysis-options>
```

Goanna allows various configuration options to fine tuned using analysis options, such as:

- Selecting what *check(s)* are enabled.
  A *check* is a Goanna term for a rule that describes a single type of potential issues that may be found within a project. For example, `ARR-inv-index` is a Goanna check that detects attempts to read a value from an array with an invalid (out-of-range) index.
  By default, Goanna uses built-in "default" set of checks which contains rules for many common issues. Using the checks selection options allows you to choose exactly which checks, or rules, that Goanna should look for. See 5.2 for more details.
- Whether to analyse the content of system and/or user header files during the analysis.

4.2.1 Running Goanna project analysis on a subset of the project

The build process that is traced using `goanna-trace` may have also collected unnecessary build information, such as third party libraries, or code related to dynamic testing.

It may be desirable to perform project analysis on only the files that are part of your project. This can be achieved by using either:

- `path` arguments, or
- `--exclude` options, or
- `--match` options.

Using the `--exclude` option

If `--exclude` options are used, then the analysed set of files must not match any of the patterns given to the `--exclude` options. For example, the command:

```
goanna-analyse --exclude=thirdparty
```

Will exclude any file that has "thirdparty" as part of its project relative path. This includes:

- `thirdparty/sqlite3/sqlite3.c`, or
- `libraries/thirdparty/xerces-c/xerces.c`, or
- `other/path/thirdparty/file.c`.

This will not exclude the following files because path patterns need to match full directory or file names:

- `thirdparty.c`, nor
- `library/thirdparty.c`, nor
- `library/thirdparty-sqlite3/sqlite3.c`.

If a path pattern contains a path separator ('/' on linux or '\' on windows), then the path pattern will match consecutive directory or file names. For example, the command:

```
goanna-analyse --exclude=thirdparty/sqlite3
```

Will exclude the following files:

- `thirdparty/sqlite3/sqlite3.c`,
- `libraries/thirdparty/sqlite3/file.c`, and
- `libraries/thirdparty/sqlite3/other.c`;

but it will not exclude:

- `athirdparty/sqlite3/file.c`, nor
- `thirdparty/sqlite3-3.12.1.1/sqlite3.c`.

Path patterns my also contain '*', '^', and '?' characters with the following effects:

'?' Match any single character

'*' Match any sub-sequence of characters, including the empty sequence.

'^' If this is used as a prefix, it means that path-patterns will only match from the beginning of the file path. For example, the command:

```
goanna-analyse --exclude=^thirdparty/sqlite3
```

will exclude `thirdparty/sqlite3/sqlite3.c`, but will not exclude: `libraries/thirdparty/sqlit3/sqlite3.c`.

Using the `--match` option

The `--match` option works in the same way as the `--exclude` option, except that only files that match the path pattern will be included in the analysis.

Using `path` arguments

`Path` arguments have the same effect as `--match` options, however they are only matched on paths exactly.

For instance, the command:

```
goanna-analyse src
```

Will analyse all files in the `src` directory, but none of the files in the `lib/src` directory.

4.3 Reading Analysis Results

The results of Goanna analysis will be made available in several places:

- During analysis, Goanna will show the name of the file(s) being analyzed, and any issues found within that file to the console; see 4.3.1 for details.
- After analysis is complete, `goanna-report` can be used to generate reports in several formats:
  - HTML warnings and summary report (see 4.3.2)
  - XML warnings report (see 4.3.3)

4.3.1 Goanna Output On The Console

The simplest way to see the result of the analysis is the output message on the console (`stderr` by default). The following is an example of the output message from Goanna:

```
Goanna - analyzing file foo.c
Number of functions: 20
foo.c:200 warning: Goanna[MEM-free-no-alloc] Severity-Medium, Pointer
    'tmp' is freed without being allocated memory.
foo.c:211 warning: Goanna[ATH-cmp-float] Severity-Low, Comparison with
    a float using == or !=.
    CERT-FLP06-C,CERT-FLP35-CPP,MISRAC2004-13.3,MISRAC++2008-6-2-2
foo.c:222 warning: Goanna[ATH-cmp-float] Severity-Low, Comparison with
    a float using == or !=.
    CERT-FLP06-C,CERT-FLP35-CPP,MISRAC2004-13.3,MISRAC++2008-6-2-2
```

Any issues in the project found by Goanna are called *warnings* (or *Goanna warnings*). In this example, Goanna shows three warnings, one from `MEM-free-no-alloc` check and two from `ATH-cmp-float` check.

By default, Goanna shows the following information for each warning:

- File name, and line number of the warning.
- Name of the check which generated this warning.
- Severity of the warning, which is one of: Low, Medium and High. Severity is determined on per-check basis and is not affected by the actual code analyzed.
- Warning message, explaining why this warning is shown.
- A list of Goanna recognized C/C++ standard rules that may be violated with this warning.

For example, in the above example, the following output message of the first warning from `ATH-cmp-float`:

```
foo.c:211 warning: Goanna[ATH-cmp-float] Severity-Low, Comparison with
    a float using == or !=.
    CERT-FLP06-C,CERT-FLP35-CPP,MISRAC2004-13.3,MISRAC++2008-6-2-2
```

Indicates that:

- On `foo.c` line 211, there is a comparison on a variable or a constant value of a type `float` or `double` using equality operators,
- This warning is determined by Goanna to have Low severity,
- This warning comes from a Goanna check called `ATH-cmp-float`, and
- The presence of this warning may mean that the project does not satisfy requirements imposed by one or more of the following rules:
  - CERT C Secure Coding Standard, Rule FLP06-C
  - CERT C++ Secure Coding Standard, Rule FLP35-CPP
  - MISRA-C:2004 Guideline, Rule 13.3
  - MISRA C++:2012 Guideline, Rule 6-2-2

It is possible to customize the format of the output message with **`--output-format`** option. See the `goanna-analyse` reference (11.3) for details about this option.

Warning Traces

Goanna also has an ability to show a trace through the execution path leading to a warning; this is called a *warning trace*. This is useful to identify important events during the execution and understand why the warning was issued.

The following is an example of warning output with a trace:

```
foo.c:254: warning: Goanna[PTR-null-fun-pos] Severity-High, Function
    call 'get_obj(o)' is immediately dereferenced, without checking for
    NULL.  CERT-EXP34-C,CWE-476
     252: ^ - if (flags & SOME_CONDITION) is false
     254: ! - possible_null
     254: > -   Entering into get_obj
     315: ^ -     if (!o) is false
     316: ^ -     switch (o->o_ty)
     320: ! -       Return NULL
```

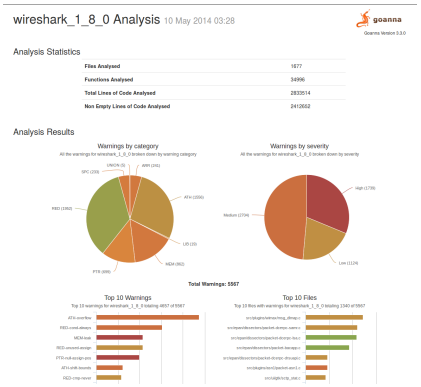
To display traces for warnings, pass **`--trace`** option to `goanna-analyse` when running analysis.

4.3.2 HTML Report of Analysis Results

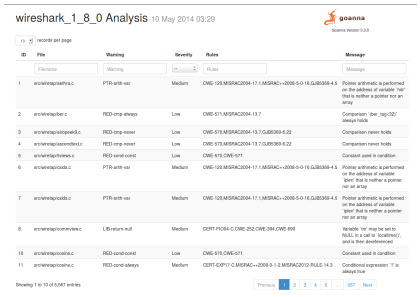
Goanna can generate an HTML report of the analysis results. There are two types of HTML reports:

- *Summary Report*, which shows a summary of the analysis result, including:
  - Basic statistics, such as the number of files analyzed,
  - Per-category and per-severity warnings pie chart,
  - Bar charts of the top 10 warnings per check and top 10 files with warnings, and
  - List of warning numbers per check.

  This report shows the high-level statistics of the analysis results in an easy-to-see format; and
- *Warnings Report*, which shows a table of all warnings found; this table is interactive and supports filtering.



Example of Summary Report



Example of Warnings Report

To generate HTML report files, use the `goanna-report` command, with the `warnings` subcommand.

`goanna-report warnings`

```
Found project name: example
Created new report directory: /example/.goanna/reports/2015-01-01_00-00
...
Generated Warnings HTML: .goanna/reports/2015-01-01_00-00/warnings.html
Generated Summary HTML: .goanna/reports/2015-01-01_00-00/summary.html
...
Created latest report directory: /example/.goanna/reports/latest
...
Copied warnings.html to latest report directory: .goanna/reports/latest
Copied summary.html to latest report directory: .goanna/reports/latest
...
```

4.3.3 Analysis Results In XML File

Goanna can generate XML reports of the analysis results. This is useful if you need Goanna to be used in conjunction with some other platform or framework and need programmatic access to the analysis result.

To generate XML report files, use the `goanna-report` command with the `warnings` subcommand.

`goanna-report warnings`

```
Found project name: example
Created new report directory: /example/.goanna/reports/2015-01-01_00-00
...
Generated Warnings XML: .goanna/reports/2015-01-01_00-00/warnings.xml
Generated Warnings File:
    .goanna/reports/2015-01-01_00-00/warnings.goannawarnings
...
Created latest report directory: /example/.goanna/reports/latest
...
Copied warnings.xml to latest report directory: .goanna/reports/latest
...
```

`goanna-report warnings` will generate the following files:

- `warnings.goannawarnings`: XML file of the last run Goanna analysis results.
- `warnings.xml`: Legacy XML output of the last run Goanna analysis results, using the old XML format that was used in Goanna 3.5.1 and before.

4.3.4 Using Goanna Dashboard Web Interface To Interact With Analysis Results

Goanna comes with the Goanna Dashboard, a web-based interface that allows you to read and interact with the analysis results.

To upload analysis results to the Goanna Dashboard, use the `goanna-snapshot` command.

`goanna-snapshot`

```
goreporter: Taking snapshot
goreporter: Project ID: 1
goreporter: Snapshot ID: 1

goreporter: To browse a snapshot of this analysis run, go to:
goreporter:
    http://localhost:1197/index.html#/?project_id=1&snapshot_id=1#snapshot
```

For more details about this feature, see Section 9.

4.4 Running Goanna Analysis using pre-3.4.0 tools

The analysis steps out-lined in this document describe the preferred tools that were released in Goanna Central 3.4.0. If you have projects that you were analysing with an older version of Goanna Central and you wish to continue using those tools, please refer to the deprecated userguide:

http://redlizards.com/resources/user-manuals/

5 Configuring Goanna Analysis

5.1 Checks Packages

Goanna Central provides over 180 individual checks, out of the box, that can be used to analyse your C/C++ source code. It is also possible to enable an additional 450+ checks that are dedicated to providing coverage for popular C/C++ standards. Controlling which checks are available to the current project is done by using the `goanna-package` command. When additional packages are enabled, the checks they provide can be selected for analysis by using the check selection options described in Section 5.2.

Goanna Central ships with the following checks packages:

- **`stdchecks`** (enabled by default): "Goanna Core" checks. This checks package contains a set of checks for common C/C++ issues.
- **`security`**: Checks that are targetting well known security vulnerabilities.
- **`misrac2004`**: Dedicated checks for MISRA-C:2004 coding standard.
- **`misrac++2008`**: Dedicated checks for MISRA C++:2008 coding standard.
- **`misrac2012`**: Dedicated checks for MISRA C:2012 coding standard.

Notes

- Goanna does not provide an option to use checks package for a single analysis run only; if you wish to do this, you must first enable a checks package, run analysis, and then disable the checks package.

5.1.1 Listing Checks Packages

To see which checks packages are installed or available, run:

```
goanna-package list
```

This will give you the list of installed and enabled checks packages, for example:

```
Installed packages:
    misrac++2008
    misrac2004
    misrac2012
    security
    stdchecks
```

If the command is run from within a project (or one is specified using the `--goanna-dir` option), then you will see additional output:

```
Packages enabled for this project:
    stdchecks
```

5.1.2 Enabling Available Checks Package

To enable an already installed checks package, run:

```
goanna-package enable-pkg <package-name>
```

Note: Use this command only for checks packages that are already installed. If you wish to enable a checks package that is not installed yet, see 5.1.4 for details.

5.1.3 Disabling Installed Checks Package

To disable an enabled checks package, run:

```
goanna-package disable-pkg <package-name>
```

5.1.4 Installing Custom Checks Package

In addition to the pre-installed checks packages, it is possible to install a custom checks package that is provided in a `.goannapackage` file.

To install and enable a custom checks package, run:

```
goanna-package install <package-file>
```

Contact Red Lizard Software for more information about custom checks packages.

Notes

- Installing a custom checks package requires write access to the installation directory of Goanna. Generally, this means:
  - On Windows, you need to issue checks package commands from within Command Prompt opened as Administrator.
  - On Linux, if you have installed Goanna Central with `sudo` or from within root terminal (i.e. under root privilege), then you need to issue checks package commands with `sudo`, or from within root terminal.

5.2 Setting Checks

5.2.1 Introduction

Goanna provides a set of options to configure what *checks*, or rules, to be used in the analysis to control types of issues that Goanna should look for.

By default, Goanna runs analysis with the default set of checks which is designed to find many common issues for most domains of application in the short time.

Goanna checks are identified by names such as `ATH-div-0` and `MEM-leak-alias`. Refer to the *Goanna Reference Guide* for the list of available and default checks, and their full description and code examples.

5.2.2 Selecting Checks With Command Line Options

To specify a set of checks to be used during analysis, pass **`--check=<check-name>`** to `goanna-analyse` when running analysis. For example:

```
goanna-analyse --check=ATH-div-0 --check=MEM-leak-alias
```

You can pass this option multiple times to specify a set of multiple checks.

Selecting All Checks Within The Coding Standard

Goanna also provides **`--checks=<name-of-standard>`** option (note the extra `s`) to specify whole group(s) of checks from popular coding standards. For example, running the following command:

```
goanna-analyse --checks=misrac2004
```

causes Goanna to run analysis with all available checks for MISRA C:2004 coding standard.

It is also possible to specify a particular rule within the coding standard. For example, to run only the check(s) corresponding to MISRA C:2004 Rule 12.8, run:

```
goanna-analyse --checks=misrac2004-12.8
```

The available coding standards in this version of Goanna Central are as follows:

Standard code	Standard name
`cert`	Computer Emergency Response Team (CERT) C/C++ Coding Standard
`cwe`	Common Weakness Enumeration (CWE)
`misrac2004`	Motor Industry Software Reliability Association (MISRA) C:2004
`misrac++2008`	Motor Industry Software Reliability Association (MISRA) C++:2008
`misrac2012`	Motor Industry Software Reliability Association (MISRA) C:2012

In addition to the above list, there is a special "standard" called **`default-on`** which, if specified, will enable the Goanna default set of checks.

Like `--check` option, you can pass `--checks` option multiple times to specify a set of multiple coding standards or coding standard rules. It is also possible to pass a mix of `--check` and `--checks` options to select all of chosen checks, coding standards and coding standard rules.

5.2.3 Enabling All Available Checks

Generally, you should only enable Goanna checks that are relevant to your environment.

Goanna analysis engine is designed to perform some computationally expensive checking algorithm only when required by the enabled checks; if you have unnecessary checks turned on, this may cause the analysis to take substantially longer.

In addition, some Goanna checks are either syntactic or specialized to particular rules within coding standards. If you have these checks turned on, this may result in a large number of spurious warnings.

Nevertheless, Goanna provides an option to enable all available Goanna checks if this is required.

To enable all available Goanna checks, pass **`--all-checks`** to `goanna-analyse` when running analysis. For example:

```
goanna-analyse --all-checks
```

Note: Using `--all-checks` option will cause any `--check`, `--checks` and `--checks-file` options to be ignored.

5.2.4 Selecting Checks for a C/C++ Project

Information about which checks are enabled for a project can be persisted using the Goanna project data store.

The `goanna-package` command can also be used to enable/disable checks:

```
goanna-package enable-check <check-name>
```

This will enable *<check-name>* for all subsequent project analyses that do not have another check related option. For example, there may be a desire to make sure a C file has not been #included by another C file. This check is called `INCLUDE-c-file`, and it is not enabled by default. To enable it for all subsequent analysis runs:

```
goanna-package enable-check INCLUDE-c-file
```

```
goanna-package: enabled check INCLUDE-c-file for current project
```

Now when **`goanna-analyse`** is next run, `INCLUDE-c-file` warnings will also be reported, in addition to the checks that are enabled by default.

```
goanna-analyse
```

```
Goanna - analyzing file foo.c
foo.c:3: warning: Goanna[INCLUDE-c-file] Severity-Low, A .c file
    shall not include any .c file.
```

5.3 Including Headers Into Analysis

By default, Goanna analyzes source files only; any header files included are read by Goanna, but are not analyzed.

To include user headers (generally, those included using `#include "..."` syntax) into analysis, pass **`--user-headers`** option to `goanna-analyse` when running analysis. For example:

```
goanna-analyse --user-headers
```

To include system headers (generally, those included using `#include <...>` syntax, such as C and C++ standard header files) into analysis, pass **`--system-headers`** option to `goanna-analyse` when running analysis. For example:

```
goanna-analyse --system-headers
```

Please note that using these options may increase time needed to run analysis.

5.4 Setting Analysis Timeouts

By default, Goanna has a timeout of 240 seconds to spend in each analysis phase within one source file. To change the timeout, pass **`--timeout=<timeout-in-seconds>`** to `goanna-analyse` when running analysis. For example:

```
goanna-analyse --timeout=60
```

Generally speaking, increasing timeout may result in more accurate results, but will take longer to complete the analysis. Decreasing timeout will improve the running time, but may result in less accurate results. Due to the underlying technology of the Goanna analysis engine, this timeout is essential.

5.5 Ignoring Certain Warnings ("Warning Suppression")

If you run Goanna frequently on the same project, in some cases, you may wish to ignore some warnings that do not require immediate attention.

Goanna provides a mechanism to allow you to manually specify such warnings to be ignored; these warnings will then not be reported in subsequent analysis. This is called *warning suppression*.

The recommended way to specify warnings to be suppressed is the Goanna Dashboard. See 9.2 for more details.

5.6 Using Knowledge Base To Guide Analysis

In some circumstances, Goanna may generate less accurate analysis results because programs may call external APIs whose source code is not avilable for Goanna analysis, or Goanna may derive inaccurate properties from source code due to limitations of static analysis. To increase analysis accuracy in these cases, Goanna provides `goanna-assume` command to import user-defined assumptions (called *knowledge base*) about source code and/or external APIs to guide Goanna analysis. To use a knowledge base when analysing your project, you must import a Goanna knowledge base file. This can be done by:

```
goanna-assume import goanna.knowledge.base.file
goanna-analyse
```

`goanna-assume` also supports exporting Goanna analysis result to Goanna knowledge base files which can be imported by Goanna later, so that users can:

- check whether Goanna has derived accurate properties about their code, make correction and re-import to override Goanna's default behaviour when needed, or
- speed up their analysis by analysing a shared library for once and reuse the exported analysis results multiple times without repeatedly analysing the library code.

To export Goanna analysis results to a knowledge base file, simply run:

```
goanna-analyse
goanna-assume export goanna.knowledge.base.file
```

`goanna-assume` also has a reset command, to clear previously imported assumptions. This will restore the following Goanna analysis to its default behaviour. To do this:

```
goanna-assume import goanna.knowledge.base.file
...
goanna-assume reset
goanna-analyse
```

Goannaknowledge base files are in a simple JSON format which is easy to read and modify. For example, the following Goanna knowledge base file specifies that external function `ex_fun` may allocate memory, and function `foo` of class `cls` in file `/home/user/project/source.cpp` does not dereference its first parameter without a `NULL` test and its return value is between -2 and 0, inclusive.

```
{
  "includes": [{ "fn": "ex_fun", "pos": -1, "pty": "alloc", "val": "maybe" }],
  "/home/user/project/source.cpp": [
    { "fn": "cls::foo", "pos": 0,  "pty": "null-deref",     "val": "false"  },
    { "fn": "cls::foo", "pos": -1, "pty": "value-interval", "val": "[-2,0]" }
  ]
}
```

Goanna provides a rich set of pre-defined properties for building assumptions about source code.

Please see 11.8 for detailed useage of `goanna-assume`.

5.7 Other Configuration Options

In addition to those listed above, Goanna provides a range of analysis options to fine tune the analysis. For the list of all analysis options, see the `goanna-analyse` reference (11.3).

Some analysis options may refer to advanced concepts not explained here; see 10 for details.

6 Running Goanna Analysis From Within IDEs

Goanna Central supports a limited form of IDE integration with the following IDEs:

- IAR Embedded Workbench version 5.40 and higher
- Keil μVision version 4 and higher

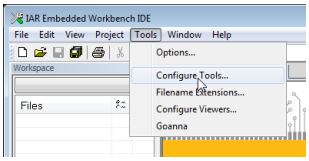
If you use one of the above IDEs, you can configure the IDE to allow Goanna to run analysis from within the IDE. This provides a convenient method to perform analysis while working on the project, without falling back to the command line.

Note: Goanna Central does not provide full IDE integration, in the sense that it does not provide a full GUI to configure analysis. Such configurations generally need to be done through command line options, similar to command line analysis.

6.1 Running Goanna Analysis From IAR Embedded Workbench®

To set up IAR Embedded Workbench IDE to trace and analyse a Embedded Workbench project, follow these steps:

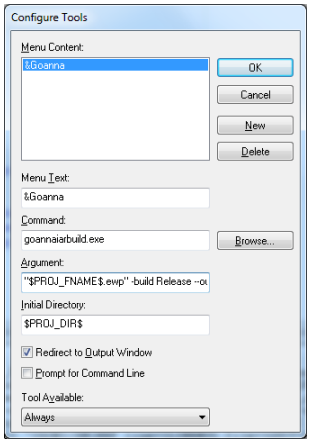
1. Start IAR Embedded Workbench.
2. From the Tools menu, select Configure Tools....



3. In the Configure Tools dialog that follows, click the New button.
4. For the Menu Text of the new menu item, type `Goanna &Init`.
5. Next to Command:, type `goanna-init.exe`.
6. Leave Arguments: empty.
7. Next to Initial Directory:, type `$PROJ_DIR$`
8. Check Redirect to Output Window.
9. Click OK.
10. In the Configure Tools dialog that follows, click the New button.
11. For the Menu Text of the new menu item, type `Goanna &Trace`.
12. Next to Command:, type `goanna-trace.exe`.
13. Next to Arguments:, copy and paste the following text:

    ```
    --force-tracing -- iarbuild.exe "$PROJ_FNAME$.ewp" -build $CONFIG_NAME$
    ```

    Note: iarbuild.exe must be in your PATH. Otherwise, please specify the full path to `iarbuild.exe`.
14. Next to Initial Directory:, type `$PROJ_DIR$`
15. Check Redirect to Output Window.
16. Click OK.



17. In the Configure Tools dialog, click the New button again.
18. For the Menu Text of the new menu item, type `Goanna &Analysis`.
19. Next to Command:, type `goanna-analyse.exe`.
20. Next to Arguments:, copy and paste the following text (make sure you remove any accidental line breaks):

```
"--output-format=%SOH%%RELFILE%:%LINENO%%SOH%: warning: Goanna[%CHECKNAME%]
Severity-%SEVERITY%, %MESSAGE%. %RULES%%EOL%"
```

21. Next to Initial Directory:, type `$PROJ_DIR$`
22. Check Redirect to Output Window.
23. Click OK.

You should now see a new menu items Goanna Init, Goanna Trace, and Goanna Analysis in your Tools menu. Click the Goanna Init menu entry to create a Goanna project store for the following Trace and Analysis commands. You should only need to run this tool once per project.

Clicking the Goanna Trace menu entry runs a full build, while collecting the compilaton processes and recording them for later analysis. You will only need to use Goanna Trace when the details of your project change, such as adding/removing a file, or changing compilation options.

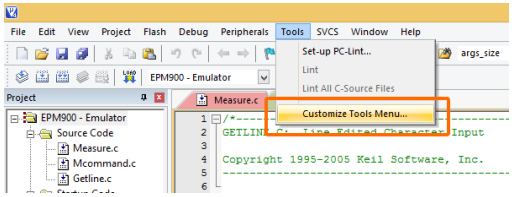
Click the Goanna Analysis menu entry will then run a project analysis, with the results appearing in the Output window.

6.2 Running Goanna Analysis From Keil™ μVision®

Note: This procedure works only for a single project. If you use this method on a multi-project workspace, then the analysis will be performed only on the active project.

To set up Keil μVision IDE, follow these steps:

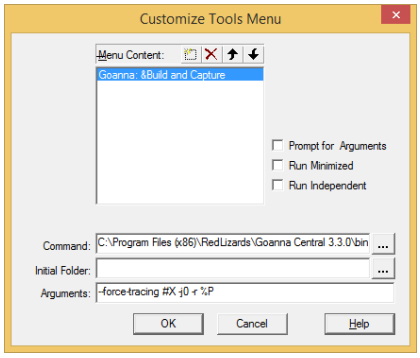
1. Start Keil μVision.
2. From the Tools menu, select Customize Tools Menu....



3. In the Customize Tools Menu dialog that follows, click the New (Insert) button. This will create a new menu entry to run `goanna-trace` to capture the build.
4. Type `Goanna: &Initalise Project` and press Enter. This will become the name of the new menu entry. You can assign different name here if you wish.
5. Click "Browse" button next to Command entry. A file selection dialog will open. Navigate to the directory where Goanna Central is installed, and choose `goanna-init.exe`.
6. Leave the Arguments: field blank
7. Click the New (Insert) button again to create a new menu entry to run `goanna-analyse` to run the analysis.
8. Type `Goanna: &Build and Capture` and press Enter. Again, you can assign different name here if you wish.
9. Click "Browse" button next to Command entry. A file selection dialog will open. Navigate to the directory where Goanna Central is installed, and choose `goanna-trace.exe`.
10. Next to Arguments:, copy and paste the following text:

```
--force-tracing -- #X -j0 -r %P
```

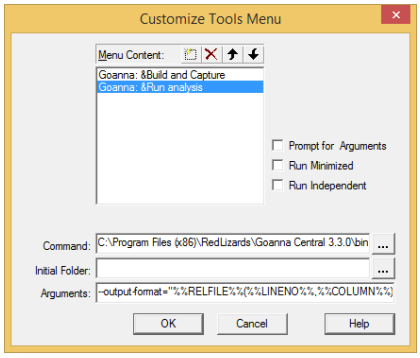
Note: If your project requires additional Keil μVision options to build, then you will need to append these here. See Description section above.



11. Click the New (Insert) button again to create a new menu entry to run `goanna-analyse` to run the analysis.
12. Type `Goanna:  &Run analysis` and press Enter. Again, you can assign different name here if you wish.
13. Click "Browse" ... button next to Command entry. Navigate to the directory where Goanna Central is installed, and choose `goanna-analyse.exe`.
14. Next to Arguments:, copy and paste the following text (make sure to remove any line breaks):

    ```
    --output-format="%%RELFILE%%(%%LINENO%%,%%COLUMN%%):  %%TYPE%%:  Goanna[%%CHECKNAME%%
    Severity-%%SEVERITY%%, %%MESSAGE%%.  %%RULES%%"
    ```

    Note: You will need to append Goanna options here to change analysis configurations. See Running analysis from Keil μVision section below.



15. Click OK.

You should now see a new menu item Goanna: Initalise Project, Goanna: Build and Capture and Goanna: Run analysis on your Tools menu (unless if named differently).

Running analysis from Keil μVision

After setting up your Keil μVision, you can follow these steps to run the analysis:

1. If it is the first time running Goanna analysis on this project, select the Goanna: Initalise Project menu item
2. Before running analysis, make sure to save any changes to the project or any source files within.
3. You can proceed to the next step (skipping the build specification generation) if your project meets all of the following conditions:
   - You have run the analysis on the project before (i.e. there is an existing build specification file)
   - Since the last analysis, there are no newly added or removed files
   - Since the last analysis, there are no changes to the project settings

   Otherwise, you will need to build the project and generate the build specification first. To do so, follow these steps:

(a) Make sure that your desired target is selected. To select a different target, you can use the drop-down selection in the Keil μVision's toolbar.

(b) Go to Tools menu and click Goanna: Build and Capture. This will rebuild your project and generate the build specification.

(c) A console window appears, and the build starts. Wait until this windows is automatically closed; when the window is closed, the build is complete.

4. To run the analysis with default settings (or custom settings saved from the last analysis - see below), go to Tools menu and click Goanna: Run analysis.

5. Alternatively, you can change the analysis option before running the analysis. To do so, follow these steps:

   (a) Go to Tools menu and click Customize Tools Menu....

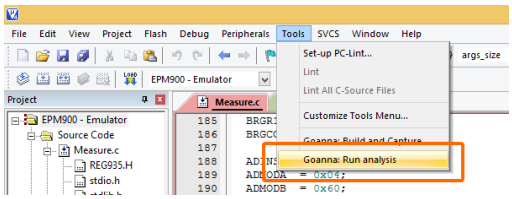
   (b) Select Goanna: Run analysis from the menu list.

   (c) Add any extra Goanna options you would like to add at the end of Arguments: text box. You can also remove options that are no longer needed. Changes made here will be saved for any subsequent analysis.

   Tips: If you frequently change the analysis options, you can enable Prompt for Arguments option to make Keil μVision ask for Goanna analysis options every time.

   (d) Press OK.

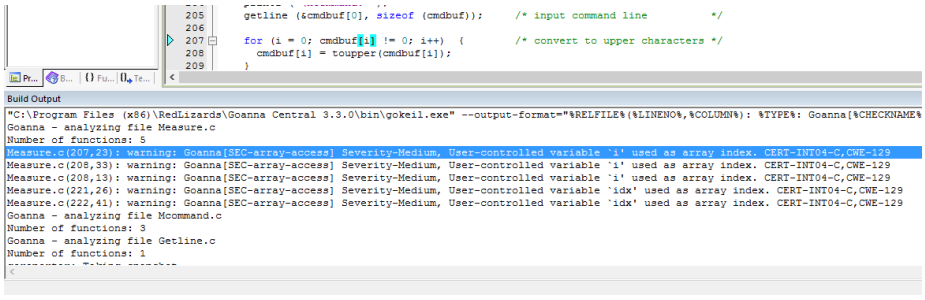
   (e) Now open Tools menu again and click Goanna: Run analysis to run the analysis.



6. Once the analysis starts, a console window appears. Wait until this window is automatically closed; when the window is closed, the analysis is complete. Analysis results should be shown in the "Build Output" window.

   Note: `goanna-analyse` will not show any progress information during analysis. This is due to technical limitation of Keil μVision.

7. You can jump to the line with a Goanna warning by double clicking at the Goanna warning line in the "Build Output" window.


getline (&cmdbuf[0], sizeof (cmdbuf)); /* input command line */
for (i = 0; cmdbuf[i] != 0; i++) { /* convert to upper characters */
cmdbuf[i] = toupper(cmdbuf[i]);
Build Output
"C:\Program Files (x86)\RedLizards\Goanna Central 3.3.0\bin\gokeil.exe" --output-format="%RELFILE%(%LINENO%,%COLUMN%): %TYPE%: Goanna[%CHECKNAME%
Goanna - analyzing file Measure.c
Number of functions: 5
Measure.c(207,23): warning: Goanna[SEC-array-access] Severity-Medium, User-controlled variable `i' used as array index. CERT-INT04-C,CWE-129
Measure.c(208,33): warning: Goanna[SEC-array-access] Severity-Medium, User-controlled variable `i' used as array index. CERT-INT04-C,CWE-129
Measure.c(208,13): warning: Goanna[SEC-array-access] Severity-Medium, User-controlled variable `i' used as array index. CERT-INT04-C,CWE-129
Measure.c(221,26): warning: Goanna[SEC-array-access] Severity-Medium, User-controlled variable `idx' used as array index. CERT-INT04-C,CWE-129
Measure.c(222,41): warning: Goanna[SEC-array-access] Severity-Medium, User-controlled variable `idx' used as array index. CERT-INT04-C,CWE-129
Goanna - analyzing file Mcommand.c
Number of functions: 3
Goanna - analyzing file Getline.c
Number of functions: 1

7 Getting the Best Results from Goanna

7.1 Interprocedural Analysis

Goanna's interprocedural analysis propagates information about function behaviour to other functions. This information includes parameter values, return values, and function effects that may impact other parts of the code. This enables Goanna to detect things in your program such as freeing of memory through function calls, functions that never return, and input values to some functions.

Interprocedural analysis is not limited to a specific set of checks, but rather enhances the precision of many checks.

An example of what interprocedural analysis can find can be seen in the sample of function `myAlloc`.

```
void *myAlloc(int param){
  void *p = malloc(param);
  if (p)
     return p;
   else
     return NULL;
}

int main(int argc, char ** argv) {
  int * n;
  n = (int*)myAlloc(sizeof(int) * 10);
  n[0] = 5; // this may be a dereference of NULL
  return *n;
}
```

Here, Goanna learns that `myAlloc` may return `NULL`. This means that when the return value of `myAlloc` is assigned to `n`, Goanna knows this value may be `NULL`. Therefore, the expression `n` may be dereferencing a `NULL` pointer, and Goanna will warn accordingly.

There is some additional computation overhead in running interprocedural analysis. If you need rapid results without much depth, then turning off interprocedural analysis will provide faster results (at the cost of accuracy in some checks). To turn off interprocedural analysis, use the `--no-ipa` option.

By default interprocedural analysis does two passes (in optimized order) over each file. This provides a good approximation for function behaviours, but may miss some complex behaviours that require many passes to accurately detect. Additional precision can be gained by increasing the iteration limit (the maximum number of passes Goanna will do). To change the interprocedural analysis iteration limit, use the `--ipa-iterations` option.

7.2 A Word on False Positives

Goanna considers all possible execution paths in your program, and will warn you if it finds potential defects (such as use of an uninitialised variable) that occur only on particular execution paths and not others. But sometimes, the execution path leading to a potential defect is actually not possible when the program is executed. If Goanna is able to deduce this through static analysis, then it won't warn you. But if it can't, then you may receive a spurious warning for a defect that isn't really there. Such warnings are called *false positives*.

Some false positives occur because Goanna currently does not track dependencies between variables in loops. For example, if you have a loop with two counters and only test one:

```
char buffer[11];
int i, count;

i = 0;
count = 10;
while (--count >= 0) {
  buffer[i++] = 'x';
}
buffer[i] = '\0';
```

Goanna may issue a false positive warning because it doesn't deduce that `i = 10` when the loop terminates.

Such false positives can often be suppressed with the `assert` macro (sec. 7.4). Otherwise you can suppress false positives using the Goanna Dashboard (see 9.2).

7.3 Using the _GOANNA Preprocessor Symbol

Goanna has a built-in preprocessor definition, defined by the macro

```
#define _GOANNA 1
```

This allows code to be explicitly included in or excluded from analysis by Goanna. For example:

```
#ifdef _GOANNA
//Code only to be included while the program is being analysed
#endif
#ifndef _GOANNA
//Code not to be analyzed by Goanna
#endif
```

7.4 Using the `assert` macro

Goanna can sometimes use information provided by `assert()` to refine its analysis of numerical and pointer values. It does this by using `assert` statements as assumptions for value ranges and pointer validity.

For example, in the code below:

```
void my_fun(void) {
  int my_array[20];
  int x = rand();

  assert(x == 10);
  f(my_array[x]);
}
```

the `assert()` means that the array reference must be in-bounds, even though the index variable `x` has a randomly-assigned value. Therefore, Goanna does not issue an out-of-bounds warning.

7.5 Sample Code

A package containing a number of sample C and C++ files is available on our website. Go to http://redlizards.com/resources/example-code/ and download the `Goanna Central Sample Code` package. The files contained in this package may be useful for practicing using Goanna, or ensuring that it is working correctly.

To run analysis on this sample code package, run:

```
goanna-init
goanna-trace make
goanna-analyse
```

You will need to install `gcc` (GNU C Compiler) and make sure that `PATH` environment variable contains a path to where `gcc` is installed before running analysis on this sample code package.

8 Using Goanna To Compute Code Metrics

8.1 Introduction

Code metrics is a set of measurements on various properties of the source code, which can help you to determine the size, complexity, and quality of your project.

Goanna Central has a separate utility to compute the code metrics, called `goanna-metrics`.

8.2 Using `goanna-metrics` To Compute Metrics

First, please ensure that you have initialised a Goanna directory (3.2) and performed build tracing (3.3).

Once the build tracing is complete, run:

```
goanna-metrics
```

to calculate metrics for all files in a project.

Code metrics will appear on the console (`stdout` by default), grouped by:

- Code metrics for each file,
- Code metrics for each function in each file, and
- Code metrics for the entire project.

The following is an example of the output message from `goanna-metrics`:

```
Metrics for file mem-ptr.c

Goanna[AVGLEVEL]: mem-ptr.c: Average level of executable statements:
    1.00
Goanna[COMF]: mem-ptr.c: Comment density: 0.10
Goanna[CONST]: mem-ptr.c: Global const declarations: 0
...

Metrics for function test (file: mem-ptr.c)

Goanna[AVGLEVEL]: (mem-ptr.c, test): Average level of executable
    statements: 1.00
Goanna[CALLED]: (mem-ptr.c, test): Number of called functions: 0
Goanna[CASE]: (mem-ptr.c, test): Else and case statements: 0
...

Metrics for project

Goanna[AVGLEVEL]: Average level of executable statements: 1.35
Goanna[CASE]: Else and case statements: 4
Goanna[COND]: If and switch statements: 16
...
```

Selecting Metrics To Compute

To specify a set of metrics to be computed, pass **`--metric=<metric-name>`** to `goanna-metrics`. For example:

```
goanna-metrics --metric=LOC --metric=VG
```

Alternatively, you can enable all metrics by passing **`--all-metrics`** option:

```
goanna-metrics --all-metrics
```

8.2.1 Computing Code Metrics For Subset Of Project

You can specify which files (in the project) to compute the code metrics for, by passing either:

- source file paths as arguments,
- `--exclude` option, or
- `--match` option.

These options work in the same way as in `goanna-analyse`. See 4.2.1 for details on how these options work.

8.2.2 Code Metrics Results In XML, HTML and CSV Files

Goanna can also export the code metrics results into XML, HTML and CSV files. To generate XML, HTML and CSV output, run `goanna-report` with `metrics` subcommand after running `goanna-metrics`:

```
goanna-report metrics
```

```
Found project name: examples
Created new metrics directory:
    /home/user/examples/.goanna/metrics/2015-08-13_11-17
Generated Metrics File:
    .goanna/metrics/2015-08-13_11-17/metrics.goannametrics
Generated Metrics HTML File:
    .goanna/metrics/2015-08-13_11-17/metrics.html
Generated Metrics CSV File:
    .goanna/metrics/2015-08-13_11-17/metrics.html
Created latest directory: /home/user/examples/.goanna/metrics/latest
Copied .goanna/metrics/2015-08-13_11-17/metrics.goannametrics to latest
    directory: .goanna/metrics/latest
Copied .goanna/metrics/2015-08-13_11-17/metrics.html to latest
    directory: .goanna/metrics/latest
Copied .goanna/metrics/2015-08-13_11-17/metrics.csv to latest
    directory: .goanna/metrics/latest
```

`goanna-report metrics` will generate the following files:

- `metrics.goannametrics`: XML file of the last run Goanna code metrics results.
- `metrics.html`: HTML file of the last run Goanna code metrics results.
- `metrics.csv`: CSV file of the last run Goanna code metrics results. By default, each column in this CSV file is separated with a comma. You can use `--delimiter` option to specify an alternative separator character.

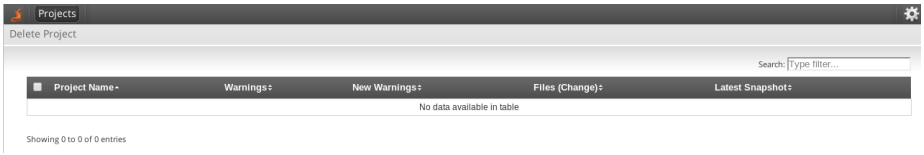
9 Using the Goanna Dashboard

Goanna Dashboard allows you to store and visualise the history of your Goanna results. It includes a web server, `goreporter`, used to display these results in a web browser. Each of your analysis runs is captured in a snapshot, which comprises of the warnings for that run as well as the source code analysed in the run. These snapshots are used to track the history of your project in the Goanna Dashboard.

9.1 Getting to the Goanna Dashboard

On Windows, and also on Linux when Goanna Central was installed under `sudo` or from within the root terminal, the Goanna Dashboard is configured to start automatically at the computer startup. In this case, you can simply navigate to http://localhost:1197/ to access the Goanna Dashboard. This will show the Project Page with all projects Goanna has taken a snapshot of.

Otherwise, you need to start the Goanna Dashboard server first. See 11.11 for instructions on how to do this.



If you load the Goanna Dashboard without first taking a snapshot you will see an empty table with the text "No data available in table". To use the Goanna Dashboard you must first take a snapshot. (refer to 11.11)

9.2 Bug Statuses

The Goanna Dashboard allows you to classify bugs into one of five statuses:

Unclassified This is the default status for when a new warning is added to the Goanna Dashboard.

Ignore Ignore this warning; useful when the warning is valid, but does not require immediate attention.

Analyse This warning need to be investigated further before it can be classified properly.

Fix This warning is a problem that needs to be fixed.

Not a Problem This is warning is not a real bug (false positive).

If you select a warning to be either **Ignore** or **Not a Problem**, then these warnings are automatically *suppressed*; Goanna will then ignore these warnings in future analysis runs.

9.3 Severity

Every check in Goanna has been assigned a static severity of either; High, Medium or Low. These severities are represented throughout the Goanna Dashboard in three colours (or a blend of these three colours):

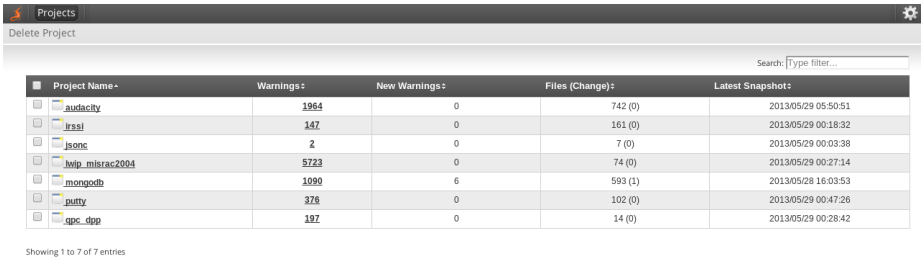
- **High:** Red
- **Medium:** Yellow
- **Low:** Green

9.4 Dashboard Views

9.4.1 Project Page

The project page is usually where you begin when using the Goanna Dashboard. It gives an overview of all projects Goanna has analysed and taken a snapshot of. The table provides a high level overview of the current state of all projects, including the number of warnings, number of new warnings, number of files analysed and the last time a snapshot was taken.

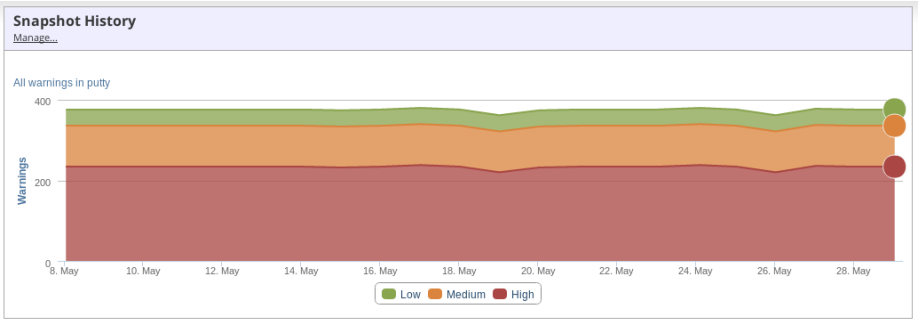
If you do not want to have a project in your dashboard anymore, you can use the checkboxes on the left hand side of the table to select the project(s) you wish to delete, then use the Delete Project button in the Dashboard's toolbar.



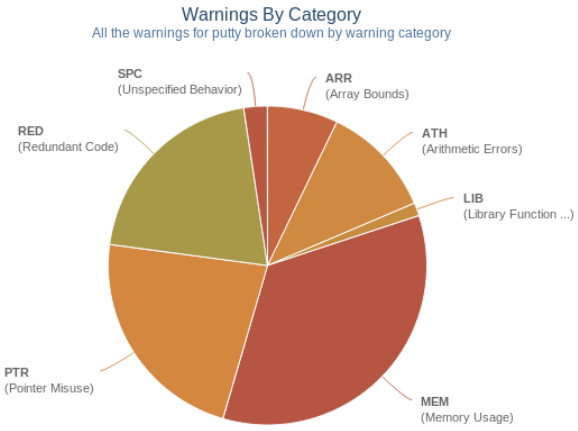
9.4.2 Report Page

Once you select a project you are taken to the project's report page. This page shows four graphs:

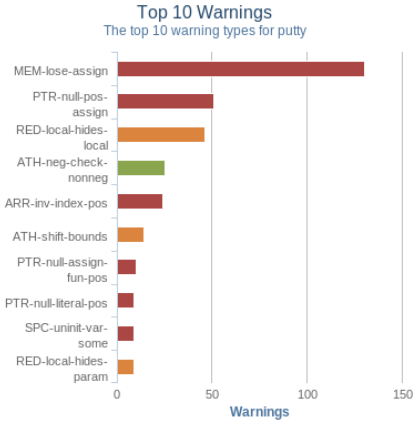
Snapshot History shows the overall progress of your project over time. Each point on the graph is one of your previous snapshots, showing the total number of warnings by severity. Clicking on any point in the graph will change the report page to show details about warnings in that snapshot. By default, the most recent snapshot is selected.



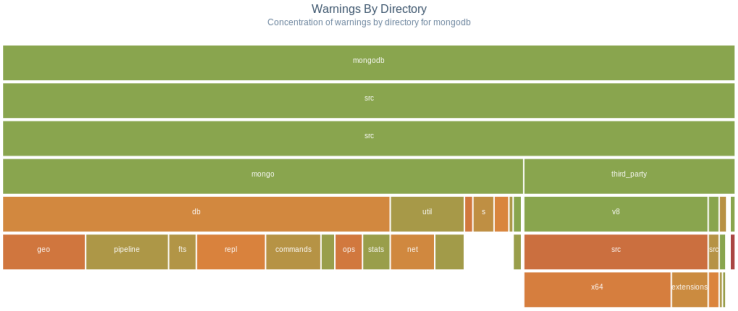
Warnings By Category shows all warnings for your snapshot broken down into each warning type. Clicking on a wedge will show the break down of warnings of that category. Clicking on a subsequent wedge will take you to the warnings browser filtered for that particular warning type.



Top 10 Warnings shows the top ten warning types in this snapshot. Clicking on one of the bars in the chart will take you to the warnings browser filtered by that warning type.

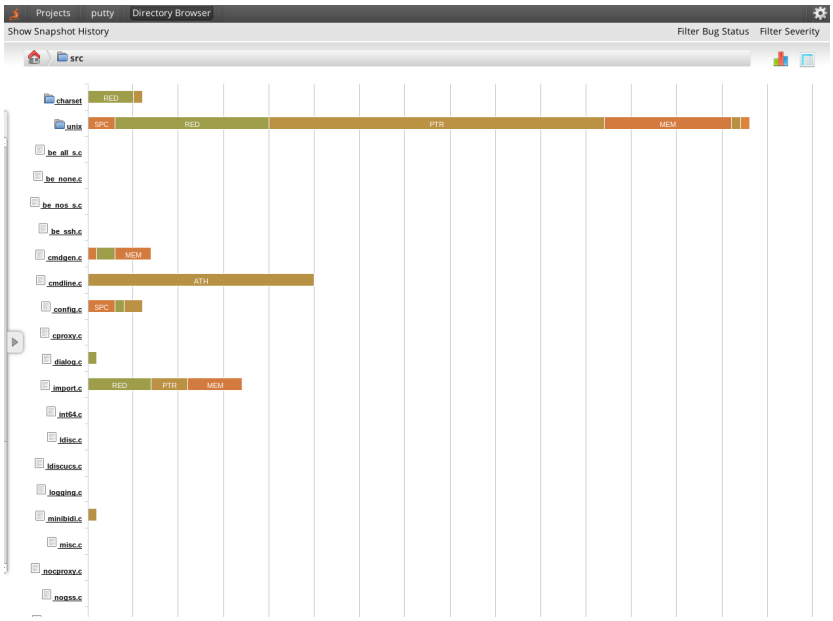


Warnings By Directory shows the concentration of warnings in your directory structure. A red node means that there are more than 10 warnings per file (average) in the directory, a green one means there are zero warnings per file (average). Clicking on a node will load the directory browser in that folder.



Global filters can be applied from the toolbar to filter by warning severity and/or bug status. These filters apply to all charts.

9.4.3 Directory Browser

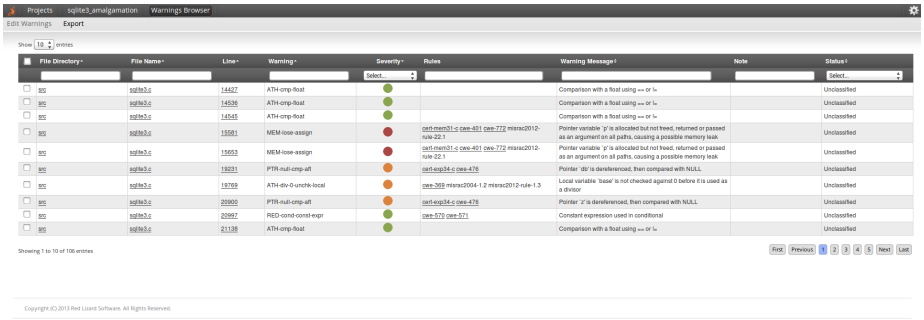


The directory browser is a way to browse through your project's directory structure to see what files and folders have what warnings. Total number of warnings per file or folder are broken down into warning categories and displayed in a bar chart.

The directory browser allows you to browse through your source tree in a few ways. The location bar above the chart allows you to see the path to your current location. Clicking on an item in the location bar will take you there in the directory browser. Similarly, clicking on a directory name in the chart will reload the directory browser with the contents of that folder. To view the contents of a source file, click on its name to load the code browser. To see the details about a particular warning category for a file, click on the segment for that category in the bar of the file or folder to load the warnings browser filtered for your selection.

The chart can be filtered in two ways. The sidebar allows for particular warning types, or warning categories to be turned on and off. In addition, the global filters in the toolbar (severity and bug status) also apply to this chart.

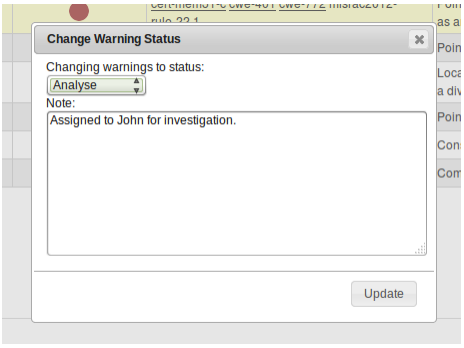
9.4.4 Warnings Browser



The warnings browser shows details of all the warnings in your project. Filtering is possible through the filter boxes in the header of the warnings table. The arrows in the table header allow for sorting.

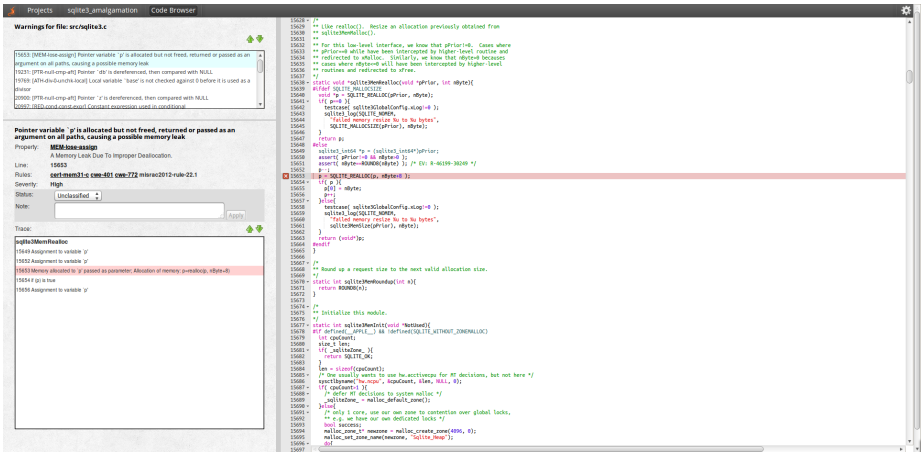
Clicking on a directory name will take you to the directory browser for that directory. A file name or line number will take you to the source code browser for that file and warning. Clicking on a Rule or Warning name will give you a description of that rule.

Selecting warnings then clicking "Edit Warnings" button opens a dialog where you can change their status and also add a note to the warnings. Clicking Update saves these changes.



The ‘Export’ button allows you to export all warnings or visible warnings to a CSV file.

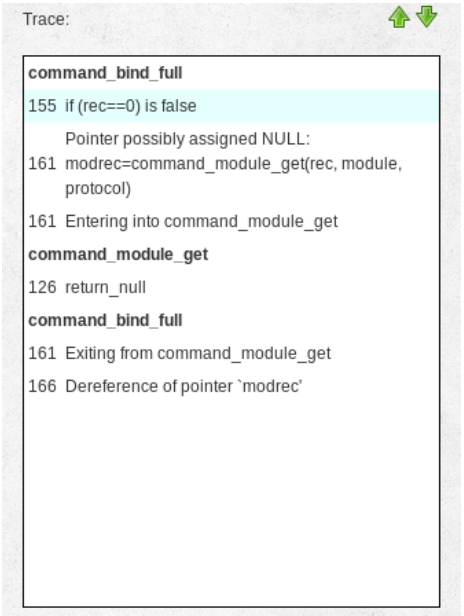
9.4.5 Code Browser



The code browser displays a file and all its Goanna warnings. The right hand side shows the source code of the file currently opened with warnings highlighted in three colours based on severity. The left hand side has the details of warnings for this file. The box at the top of the left hand pane allows you to select a warning to see in more detail.

Like in the warnings browser, you are able to change the status and add a note to each warning in the code browser. This can be done by changing the status and/or adding a note and pressing Apply.

If there is trace information for a warning it will also appear in this pane. You can step through the trace just like a debugger. To go to a step in the trace, click on it and the source code browser will jump to the corresponding line. You can then navigate through the trace using either the up/down arrows on top of the trace dialog, or by using the up/down arrow keys on your keyboard.

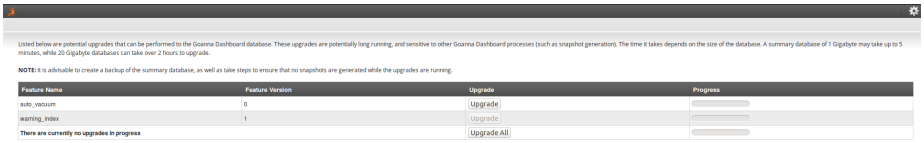


9.5 Database Upgrades

If you have upgraded from an older version of Goanna, Goanna may offer you to perform optional "database upgrades" to improve the performance of the Goanna Dashboard.

When you open Goanna Dashboard with the existing database, you may see the following notification:

Clicking the link will show a list of available optional upgrades:



In this version of Goanna, the following optional upgrades (called *features*) may be available:

- **`auto_vacuum`:** In Goanna 3.2.4 and before, deleting a snapshot or a project did not immediately delete corresponding data from the database. This may result in the database size to never shrink.

  Performing this upgrade causes Goanna to delete all residual data from removed snapshots and projects, and set up the database so that when a snapshot or a project is removed, Goanna removes corresponding data immediately. This ensures that the database size is always minimal.

  This upgrade was introduced in Goanna 3.3.0.
- **`warning_index`:** Performing this upgrade causes Goanna to apply optimizations to database indexes, resulting in improved performance, especially when loading Warnings Browser view.

  This upgrade was introduced in Goanna 3.3.0.

To apply an upgrade, click "Upgrade" button next to a desired feature. Alternatively, clicking "Upgrade All" button will apply all available upgrades.

Important Notes

We recommend that you take a backup of the database before applying any of the upgrade. The database is located in the following location:

- On Windows XP and Windows Server 2003:
  `C:\Documents and Settings\<name-of-user-who-installed-goanna>\Local Settings\Application Data\RedLizards\Goanna Central\summary.goannadb`
- On Windows Vista, Windows Server 2008, and all later versions of Windows:
  `C:\Users\<name-of-user-who-installed-goanna>\AppData\Local\RedLizards\Goanna Central\summary.goannadb`
- On Linux:
  `<installation-directory>/reporter/summary.goannadb`

You should not run Goanna analysis, interact with the Goanna Dashboard, or otherwise run any Goanna commands, until the upgrade is complete.

Performing upgrade will take a long time depending on the size of the database. If your database is significantly large, this may take hours to complete.

9.6 Project Settings (Advanced)

In general, the Goanna Dashboard should work for your project out of the box. However, there are a few advanced settings available if you want to customise the behaviour.

Settings are applied on a per-project basis. To access your project's settings click the gear menu in the top right hand corner after loading your project (through the project page), and select Project Settings.

9.6.1 Code Browser Character Encodings

By default the Dashboard will recognize source files in ASCII and UTF-8 (and additionally on Windows, the default character encoding used by your system). If your source files are not in any of these encodings, you will need to specify the encoding here in order for it to display correctly in the source code browser. A link is available in the Project Settings window listing all the encodings supported.

10 Advanced Features, Concepts and Configurations

10.1 Proceed With Caution

This section describes a number of advanced Goanna concepts and configuration options. In most cases, there is no need to know these concepts or use these options. We recommend that you use these advanced options only when required.

Many of these advanced features contain limitations and/or caveats; if you wish to use any of these features, then make sure to read the description carefully to understand these limitations and caveats.

Important: Red Lizard Software may deprecate, and later remove, any feature within this section in the future releases of Goanna.

10.2 Manually Running Analysis On Source Files

`goannacc` and `goannac++` utilities can be used to run Goanna analysis on C or C++ source file(s), without first performing build tracing.

To run Goanna analysis directly on C source files, run:

```
goannacc --nc --with-cc=<path-to-C-compiler> --db=<path-to-database> <extra-arguments-to-goannacc> <arguments-to-compiler>
```

To run Goanna analysis directly on C++ source files, run:

```
goannac++ --nc --with-cxx=<path-to-C++-compiler> --db=<path-to-database> <extra-arguments-to-goannac++> <arguments-to-compiler>
```

For example, to run analysis on `src/main.c` (when the source file is written for `gcc` compiler):

```
goannacc --nc --with-cc=gcc --db=myfiles.goannadb src/main.c
```

This will run analysis on `src/main.c`, and outputs the result to the console.

Additionally, this command stores information about the analysis into a specified *project database* file via **`--db=<path-to-database>`** option (see 10.4 for details). Passing `--db` option is essential to ensure the maximum accuracy of the analysis, and allows Goanna to cache some information to speed up subsequent analysis.

You can pass most analysis configuration options (such as `--check` option) to `goannacc` or `goannac++` to control the analysis. See 11.9 for the complete list of available options.

Note: The following features are not available if you run analysis directly on C/C++ source files:

- Knowledge base
- HTML analysis report
- XML analysis report
- Publish to Goanna Dashboard
- Publish to SonarQube

Compile and Run Analysis At The Same Time

`goannacc` and `goannac++` can also act as compiler wrappers; that is, they can be used to compile source files and perform analysis on these files at the same time.

To compile and run analysis at the same time, run the same command as above, but without **`--nc`** option. For example:

```
goannacc --with-cc=gcc --db=myfiles.goannadb src/main.c
```

This will compile `src/main.c` using `gcc`, and then performs analysis on that file.

Note: Goanna will not perform analysis in this mode if the specified files fail to compile. If this behaviour is not desirable, pass **`--ignore-errors`** option. Please note that, if the files fail to compile because of syntax errors, then Goanna analysis may fail to run, even with **`--ignore-errors`** option.

10.3 Manually Running Link Time Analysis On Object Files

Similar to source files, it is possible to perform link time analysis directly using `goannald`.

Because link time analysis relies on information in the project database about source files, you must first run source file analysis on all relevant source files before running link time analysis.

Once source file analysis is complete, run the following command to run link time analysis:

```
goannald --nc --with-ld=<path-to-linker> --db=<path-to-database> <extra-arguments-to-goannald> <arguments-to-linker>
```

For example, to run analysis on `src/main.o` object file (when using GNU Linker (`ld`)):

```
goannald --nc --with-ld=ld --db=myfiles.goannadb src/main.o
```

This will run link time analysis on `src/main.o`, and outputs the result to the console.

Note: The following features are not available if you run link time analysis directly:

- Knowledge base
- HTML analysis report
- XML analysis report
- Publish to Goanna Dashboard
- Publish to SonarQube

Link and Run Link Time Analysis At The Same Time

Like `goannacc`, `goannald` can also act as a linker wrapper to link object files and perform link time analysis at the same time.

To link and run link time analysis at the same time, run the same command as above, but without **`--nc`** option. For example:

```
goannald --with-ld=ld --db=myfiles.goannadb src/main.o
```

This will perform linking with an object file `src/main.o` using `ld`, and then performs link time analysis on that object file..

Note: Goanna will not perform link time analysis in this mode if the specified object files fail to link. If this behaviour is not desirable, pass **`--ignore-errors`** option.

10.4 The Project Database

When analyzing a whole project, Goanna stores information about the project in a database file (called *project database* or *project DB*) that is used by all the `goanna` analysis commands. The project database is also used by `goreporter` to summarize information for the Goanna Dashboard.

The project database file by convention has the extension `.goannadb`, and usually resides inside your `.goanna` directory.

Goanna stores the following information in the project database:

- Interprocedural (or whole-program) analysis information, that is, information about bugs that occur as a result of calls between functions.
- Cached information about your source files, which enables incremental analysis (that is, re-analysis of a project after small changes) to be performed much faster.
- Suppression information, if you have suppressed warnings.

You should not delete this database file unless when required. If you delete this database file, then the next analysis run will take much longer (due to cached information being removed), and you will also lose warnings suppression information.

Unlike other Goanna utilities, if you use `goannacc`, `goannac++`, or `goannald` directly to perform analysis (see 10.2), then you must explicitly specify the location of the project database with **`--db=`***<file>* option. Not specifying this option causes:

- Goanna analysis to produce inaccurate results because information needed for interprocedural analysis becomes unavailable,
- Link-time analysis to become unavailable (link-time analysis relies on the project database), and
- Warnings suppression functionality to not work.

10.5 How Goanna's Compiler Support Work

Most C/C++ code uses non-standard language extensions to some extent. Even if your own code is 100 percent standard compliant, you must almost certainly include libraries and header files that use non-standard extensions provided by the compiler.

Goanna analyzes C/C++ code at a very deep semantic level. In order to do this, it must analyze your source code exactly as understood by your compiler, including all the same system headers, built-in predefined macros, and language extensions.

In some dialects, the macros and include paths predefined in the compiler are not fixed, but vary depending on your operating system, how the compiler has been configured and built, and even on what command line options are given to it at run time.

Goanna utilities are designed to handle all these complexity automatically, so that Goanna can fully understand all of your C/C++ source files, even if they contain compiler-specific syntax extensions or include library headers in other directories.

When you perform a project-wide analysis, Goanna automatically does the following things to ensure Goanna can understand your C/C++ source files:

1. Goanna switches its C/C++ parsing mode, based on information in the build specification. Each parsing mode is called *dialect in use*,
2. Goanna detects all predefined macros, include paths, and any other built-in configuration options from various sources (usually by invoking a compiler in question, or fetching information from environment variables), and
3. Goanna reads all compiler-specific command line options and configures the parser accordingly to ensure any syntax related options are applied correctly.

If you run source file or link time analysis manually by directly calling `goannacc`, `goannac++` or `goannald`, then you need to ensure that you pass **`--with-cc=<path-to-C-compiler>`**, **`--with-cxx=<path-to-C++-compiler>`**, and **`--with-ld=<path-to-linker>`** options. These options specify which compilers and linkers your project uses, and ensures that Goanna is configured appropriately for your environment.

11 Goanna Central Utility Reference

11.1 `goanna-init` – Initialise C/C++ Projects

Synopsis

```
goanna-init [options]
```

Description

`goanna-init` creates a Goanna project data store. This file system location is used to group Goanna settings and Goanna analysis data that is associated with the project.

The project data store is used by the other Goanna commands: `goanna-trace`, `goanna-analyse`, `goanna-report`, and `goanna-package`.

Command Line Options

`--goanna-dir=<path>` Specify the location of the Goanna project data; default is `.goanna`

`--project-dir=<path>` Specify the location of the project root; default is `..`

If the `--project-dir` option is given, then `--goanna-dir` is also required.

`--project-name=<name>` Specify the name of the project. This is used as an identifier for reporting purposes (such as HTML reports). By default the project name is determined from the name of the project root directory.

`--help` Print help message for common options.

`--version` Print version information.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--advanced-help` Print help message for advanced options.

Return Codes

Here is a list of return codes `goanna-init` returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goanna-init` completed execution successfully.

- `goanna-init` completes execution without errors
- `goanna-init` prints help or version information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- The `--project-dir` option was given, but the `--goanna-dir` option was missing.
- `goanna-init` was unable to create the Goanna project data store.
- `goanna-init` encounters an error while parsing command-line arguments

11.2 `goanna-trace` – Monitor C/C++ Project Build

Synopsis

```
goanna-trace [options] [--] <build-command>
```

Description

`goanna-trace` will execute *<build command>* and record the compilation and linker processes that were executed in order to build the project.

Command Line Options

`--goanna-dir=`*`<path>`* Specify the location of the Goanna project data; default is `.goanna`

`--project-dir=`*`<path>`* Specify the location of the project root; default is `..`
If the `--project-dir` option is given, then `--goanna-dir` is also required.

`--license-server=`*`<server[:port]>`* Attempt to contact a license server at address *<server>*. *<Port>* is optional (defaults to 6200).

`--license-borrow-hours=`*`<number>`* When contacting license server, borrow license for *<number>* of hours. Defaults to 1, maximum of 24.

`--force-tracing` Force build recording even when the specified build spec file already exists.

`--` Stop processing command line options and treat subsequent options as arguments.

`--help` Print help message for common options.

`--version` Print version information.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--license-dir=`*`<directory>`* Set directory in which to look for a license file.

`--advanced-help` Print help message for advanced options.

Return Codes

`goanna-trace` returns the return code from the underlying build process.

11.3 `goanna-analyse` – Analyse C/C++ Projects

Synopsis

```
goanna-analyse [options] [<path>]
```

Description

`goanna-analyse` will analyse a C/C++ project using the build spec created by `goanna-trace` (see 11.2).

Command Line Options

`--goanna-dir=<path>` Specify the location of the Goanna project data; default is `.goanna`

`--project-dir=<path>` Specify the location of the project root; default is `..`

If the `--project-dir` option is given, then `--goanna-dir` is also required.

`--exclude=<path-pattern>` Do not analyse files that match `<path-pattern>`.

`--match=<path-pattern>` Only analyse files that match `<path-pattern>`.

`--dryrun` Instead of analysis, output which files and executables will be analysed.

`--license-server=<server[:port]>` Attempt to contact a license server at address *<server>*. *<Port>* is optional (defaults to 6200).

`--license-borrow-hours=<number>` When contacting license server, borrow license for *<number>* of hours. Defaults to 1, maximum of 24.

`--jobs=<number>` The number of parallel jobs (1-20) to run simultaneously. Default is the number of CPU cores detected on your system, note that running `goanna --help` will display the number of detected cores as the default.

`--ungroup` With `--jobs=2` or more, print all output immediately instead of grouped by job. This reduces latency and memory usage but may cause output from different jobs to be mixed.

`--force-analysis` Re-analyze files that have not changed since last run.

`--all-checks` Run all available checks (overrides all other check related options).

`--check=<name>` Run a specific check (overrides any checks file).

`--checks=<standard>` Run all checks in the specified coding standard. For example, `--checks=misrac2004` runs all available checks in the MISRA C:2004 standard.

`--checks=<standard>-<rule>` Run the check(s) corresponding to one rule in the specified coding standard. For example, `--checks=misrac2004-12.8` runs the check(s) that implement MISRA C:2004 rule 12.8.

`--no-user-headers` Do not process user header files.

`--user-headers` Process user header files. Default: no.

`--no-system-headers` Do not process system header files.

`--system-headers` Process system header files. Default: no.

`--header-protos` Process header files as if they only had prototypes. Default: no.

`--no-header-protos` Treat header files normally, instead of processing them as having only prototypes.

`--timeout=<n>` Set a timeout (in seconds) for analysis of each source file. Default: 240.

Important: Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

`--output-format=<format>` Specify a warning format used by Goanna to output warnings. The following special strings in *<format>* are expanded:

`%FILENAME%` the filename

`%RELFILE%` the filepath and filename

`%RELPATH%` the filepath

`%ABSFILE%` the absolute filepath and filename

%ABSPATH% the absolute filepath
%DBRELFILE% the filepath relative to the database file and filename
%DBRELPATH% the filepath relative to the database file
%LINENO% the line number
%COLUMN% the column number
%CHECKNAME% the check identifier
%SEVERITY% the checks severity rating
%MESSAGE% the warning message
%RULES% corresponding rule(s) from coding standards, if any
%TRACE% counter example if any
%FUNCTION% the function name
%SUPPRESSED% a * if the warning is suppressed
%WARNINGID% the hash of this warning
%EOL% a line break
%% a literal %.

The default warning format is: "`%RELFILE%:%LINENO%: warning: Goanna[%CHECKNAME%] Severity-%SEVERITY%, %MESSAGE%. %RULES%%EOL%`".

`--trace` Prints out a trace through the function that leads to the warning. This is helpful for understanding why the warning occurs.

`--trace-format=<`*format*`>` Specify the format to output traces. The following special strings are used in the trace format:

%FILENAME% the filename
%RELFILE% the filepath and filename
%RELFILEX% the filepath and filename followed by "**:**", or blank if in the current source file
%RELPATH% the filepath
%ABSFILE% the absolute filepath and filename
%ABSFILEX% the absolute filepath and filename followed by "**:**", or blank if in the current source file
%ABSPATH% the absolute filepath
%DBRELFILE% the filepath relative to the database file and filename
%DBRELPATH% the filepath relative to the database file
%FUNCTION% the function name
%LINE% the line number
%TEXT% text describing the event on the trace
%TYPE% the type of the trace line
%EOL% a line break
%% a literal %.

The default trace format is: "`%LINE%: %TYPE% - %TEXT%%EOL%`".

`--` Stop processing command line options and treat subsequent options as arguments.

`--help` Print help message for common options.

`--version` Print version information.

Note: `--exclude` options will cause errors if `--match` options, or `<path>` arguments are given.

`--match` options are allowed in conjunction with `<path>` arguments.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--license-dir=<`*directory*`>` Set directory in which to look for a license file.

`--timeout-per-phase=<n>` Set a timeout (in seconds) for each phase of analysis. This is useful if you have a few functions that take very long to analyze and you would like to limit the time spent on these, while still getting as many results as possible on everything else. Default: 60.

Important: Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

`--issue-report=<type>` Control generation of issue report files:

never Never

on-failure On failures only

on-error On failures and analysis errors

timeout On failures, errors, and timeouts

always Always (even if successful)

`--advanced-help` Print help message for advanced options.

Return Codes

Here is a list of return codes `goanna-analyse` returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goanna-analyse` completed execution successfully.

- `goanna-analyse` completes execution without errors
- `goanna-analyse` prints help or version information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- The `--project-dir` option was given, but the `--goanna-dir` option was missing.
- `goanna-analyse` was unable to access the Goanna project data store.
- `goanna-analyse` encounters an error while parsing command-line arguments

11.4 `goanna-report` – Generate C/C++ Project Reports

Synopsis

```
goanna-report [options] <warnings>
goanna-report [options] <metrics>
```

Description

`goanna-report` will produce reports in various output formats for a C/C++ project, using the analysis information created by `goanna-analyse` or `goanna-metrics` (see 11.3).

Command Line Options

`--goanna-dir=<path>` Specify the location of the Goanna project data; default is `.goanna`

`--project-dir=<path>` Specify the location of the project root; default is `..`

If the `--project-dir` option is given, then `--goanna-dir` is also required.

`--report-group=<group>` Optionally specify a group for reports to be generated in.

`--latest=<name>` Optionally specify the name of the `<latest>` copy of the current report.

`--delimiter=<character>` Specify a delimiter to be used to separate each column in CSV output (by default, this is a comma). This option has no effect when generating analysis reports (when running `goanna-report` with `warnings` subcommand).

`--help` Print help message for common options.

`--version` Print version information.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--advanced-help` Print help message for advanced options.

Return Codes

Here is a list of return codes `goanna-report` returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goanna-report` completed execution successfully.

- `goanna-report` completes execution without errors
- `goanna-report` prints help or version information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- The `--project-dir` option was given, but the `--goanna-dir` option was missing.
- `goanna-report` was unable to access the Goanna project data store.
- `goanna-report` encounters an error while parsing command-line arguments

11.5 `goanna-snapshot` – Upload C/C++ Project Snapshots

Synopsis

```
goanna-snapshot [options]
```

Description

`goanna-snapshot` will upload the analysis information created by `goanna-analyse` (see 11.3) as a snapshot into the Goanna Dashboard (see 9).

Command Line Options

`--goanna-dir=<path>` Specify the location of the Goanna project data; default is `.goanna`

`--project-dir=<path>` Specify the location of the project root; default is `..`

If the `--project-dir` option is given, then `--goanna-dir` is also required.

`--help` Print help message for common options.

`--version` Print version information.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--advanced-help` Print help message for advanced options.

Return Codes

Here is a list of return codes `goanna-snapshot` returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goanna-snapshot` completed execution successfully.

- `goanna-snapshot` completes execution without errors
- `goanna-snapshot` prints help or version information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- The `--project-dir` option was given, but the `--goanna-dir` option was missing.
- `goanna-snapshot` was unable to access the Goanna project data store.
- `goanna-snapshot` encounters an error while parsing command-line arguments

11.6 goanna-package – Customise C/C++ Project Analysis

Synopsis

```
goanna-package [options] help
goanna-package [options] help <command>
goanna-package [options] list
goanna-package [options] install <goanna package file>
goanna-package [options] enable-pkg <package name>
goanna-package [options] disable-pkg <package name>
goanna-package [options] enable-check <check name>
goanna-package [options] disable-check <check name>
```

Description

goanna-package is used to access and modify configuration options relating to which Goanna checks packages will be used by goanna-analyse. The configuration is on a per-project basis, using the Goanna project data store created by goanna-init.

Command Line Options

--goanna-dir=<*path*> Specify the location of the Goanna project data; default is .goanna

--project-dir=<*path*> Specify the location of the project root; default is ..

If the --project-dir option is given, then --goanna-dir is also required.

Return Codes

Here is a list of return codes goanna-package returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of 0 generally means that no errors were encountered, and goanna-package completed execution successfully.

- goanna-package completes execution without errors
- goanna-package prints help or version information

1 (One) A return code of 1 generally means that a user-provided parameter is invalid.

- The --project-dir option was given, but the --goanna-dir option was missing.
- goanna-package encounters an error while parsing command-line arguments

11.7 goanna-metrics – Calculate C/C++ Project Code Metrics

Synopsis

```
goanna-metrics [options] [<path>]
```

Description

goanna-metrics calculates metrics for a project by using the build spec created by goanna-trace.

Command Line Options

--goanna-dir=*<path>* Specify the location of the Goanna project data; default is .goanna

--project-dir=*<path>* Specify the location of the project root; default is ..
If the --project-dir option is given, then --goanna-dir is also required.

--exclude=*<path-pattern>* Do not analyse files that match <path-pattern>.

--match=*<path-pattern>* Only analyse files that match <path-pattern>.

--license-server=*<server[:port]>* Attempt to contact a license server at address *<server>*. *<Port>* is optional (defaults to 6200).

--license-borrow-hours=*<number>* When contacting license server, borrow license for *<number>* of hours. Defaults to 1, maximum of 24.

--jobs=*<number>* The number of parallel jobs (1-20) to run simultaneously. Default is the number of CPU cores detected on your system, note that running goanna --help will display the number of detected cores as the default.

--ungroup With --jobs=2 or more, print all output immediately instead of grouped by job. This reduces latency and memory usage but may cause output from different jobs to be mixed.

--metric=*<name>* Run a specific metric.

--all-metrics Run all available metrics.

-- Stop processing command line options and treat subsequent options as arguments.

--help Print help message for common options.

--version Print version information.

Note: --exclude options will cause errors if --match options, or <path> arguments are given.

--match options are allowed in conjunction with <path> arguments.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

--license-dir=*<directory>* Set directory in which to look for a license file.

--advanced-help Print help message for advanced options.

Return Codes

Here is a list of return codes goanna-metrics returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goanna-metrics` completed execution successfully.

- `goanna-metrics` completes execution without errors
- `goanna-metrics` prints help or version information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- The `--project-dir` option was given, but the `--goanna-dir` option was missing.
- `goanna-metrics` was unable to access the Goanna project data store.
- `goanna-metrics` encounters an error while parsing command-line arguments

11.8 goanna-assume – Import/Export Knowledge Base

Synopsis

```
goanna-assume [options] help
goanna-assume [options] help <command>
goanna-assume [options] reset
goanna-assume [options] import <goanna knowledge base file>
goanna-assume [options] export [options] <goanna knowledge base file>
```

Description

`goanna-assume` is used to import knowledge base about source code to guide Goanna analysis and also export Goanna analysis results for use in future Goanna runs.

Knowledge base imported by the `import` subcommand will be persistent for subsequent executions of `goanna-analyse`, until another Goanna knowledge base file is imported or the `reset` subcommand is executed, which will clear all previously imported knowledge base.

Previous Goanna analysis results stored in the project database can be exported in the form of Goanna knowledge base file by executing the `export` subcommand. By default, this command generates results for all functions and groups results by source file. However, it will only generate results for non-static functions when `--as-api` or `--api-list` options are used.

Important: Using `import` or `reset` subcommand causes Goanna to delete all analysis data currently stored, meaning that next analysis will be slower. If you have suppressed warnings, this information will also be deleted.

Command Line Options

`--goanna-dir=`*`<path>`* Specify the location of the Goanna project data; default is `.goanna`

`--project-dir=`*`<path>`* Specify the location of the project root; default is `..`

If the `--project-dir` option is given, then `--goanna-dir` is also required.

`--rel-path` Use project-relative path in file names.

`export`

`--as-api` Instead of grouping function analysis results by source file, export results for all non-static functions in the API section.

`--api-list=`*`<api list file>`* Only export function analysis results for functions listed in *<api list file>*.

If the `--api-list` option is given, then `--as-api` is also enabled.

File Format

A Goanna knowledge base file is a JSON file containing an associative list, with each member of the list specifying functions in a particular file. The name of the member is the file name (specified with an absolute path by default, or project-directory relative path if option `-rel-path` is used), and the value is a list of function specifications. Optionally, there can a special member (with the name `"includes"`) containing specifications for external API functions to be used in Goanna analysis.

A function specification specifies a property and its associated value for a function parameter, when the value of `"pos"` is zero-indexed parameter position, or a function's return value/side effect, when the value of `"pos"` is -1.

A property generally have a value of

`true` When users want to inform Goanna that the property is valid for the code specified.

`maybe` When users want to inform Goanna that the property may be valid for the code specified.

`false` When users want to inform Goanna that the property is not valid for the code specified.

The only exception is for interval properties, which have values in the format of **`[lb,ub]`** where `lb` and `ub` are integers representing the lower and upper bounds of the closed interval specified.

For example, the following Goanna knowledge base file specifies that external function `ex_fun` may allocate memory, and function `foo` of class `cls` in file `/home/user/project/source.cpp` does not dereference its first parameter without a null test and its return value is between -2 and 0, inclusive.

```
{
  "includes": [{ "fn": "ex_fun", "pos": -1, "pty": "alloc", "val": "maybe" }],
  "/home/user/project/source.cpp": [
    { "fn": "cls::foo", "pos": 0,  "pty": "null-deref",     "val": "false"  },
    { "fn": "cls::foo", "pos": -1, "pty": "value-interval", "val": "[-2,0]" }
  ]
}
```

Details about available propeties and their leagal values can be find in the following subsection.

The following table lists this file format in Backus-Naur Form.

```
⟨goanna-knowledge-base-file⟩  ::=  '{}'
                              |    '{' ⟨file-spec-list⟩ '}'
⟨api-list-file⟩               ::=  '[]'
                              |    '[' ⟨fn-name⟩ { ',' ⟨fn-name⟩ } ']'

⟨file-spec-list⟩              ::=  ⟨api-spec⟩
                              |    ⟨api-spec⟩ { ',' ⟨file-spec⟩ }
                              |    ⟨file-spec⟩ { ',' ⟨file-spec⟩ }
⟨api-spec⟩                    ::=  '"includes":[]'
                              |    '"includes":[' ⟨fn-spec-list⟩ ']'
⟨file-spec⟩                   ::=  ⟨file-name⟩ ':[]'
                              |    ⟨file-name⟩ ':[' ⟨fn-spec-list⟩ ']'

⟨file-name⟩                   ::=  string of file name
⟨fn-spec-list⟩                ::=  ⟨fn-spec⟩ { ',' ⟨fn-spec⟩ }
⟨fn-spec⟩                     ::=  '{"fn":' ⟨fn-name⟩ ', "pos":' ⟨pos⟩ ','
                                    '"pty":' ⟨pty⟩ ', "val":' ⟨value⟩ '}'
⟨fn-name⟩                     ::=  '"' [ ⟨namespace⟩'::' ] [ ⟨class⟩'::' ] ⟨function⟩ '"'
⟨namespace⟩, ⟨class⟩, ⟨function⟩  ::=  string
⟨pos⟩                         ::=  ⟨return-pos⟩ | ⟨param-pos⟩
⟨return-pos⟩                  ::=  -1
⟨param-pos⟩                   ::=  zero-indexed parameter position
⟨pty⟩                         ::=  '"' ⟨param-pty⟩ '"' | '"' ⟨return-pty⟩ '"'

⟨param-pty⟩                   ::=  ptr-read | ptr-written
                              |    ptr-freed | ptr-deleted | null-deref | divisor
                              |    file-closed | file-written | file-path | lib-path
                              |    relative-path | strconst | xpath-query
                              |    sql-statement | os-command | credential
                              |    format-str | strcpy-source | taint-source
                              |    taint-overwritten | ldap-query | loop-cond-var
                              |    memcpy-arg | alloc-size
                              |    array-access | buffer-offset
                              |    value-interval | array-size-interval
                              |    req-gt-z | req-ge-z | req-ne-z
                              |    req-lt-1 | req-le-1
                              |    req-gt-n1 | req-ge-n1
                              |    req-le-ff | req-gt-min
                              |    return-ec | modify-errno
⟨return-pty⟩                  ::=  relative-path | strconst | taint-source
                              |    alloc | null | chdir | chroot | impure | exit
                              |    value-interval | array-size-interval

⟨value⟩                       ::=  '"true"' | '"false"' | '"maybe"'
                              |    '"[' ⟨bound⟩ ',' ⟨bound⟩ ']"'
⟨bound⟩                       ::=  integer
```

An API list file for the `export` subcommand is also a JSON file containing a list of unmangled names for functions whose results need to be expoted. For example:

```
[ "my_alloc", "my_free", "cls::foo" ]
```

The following table lists this file format in Backus-Naur Form.

⟨*api-list-file*⟩ ::= '`[]`'
| '`[`' ⟨*fn-name*⟩ { '`,`' ⟨*fn-name*⟩ } '`]`'

⟨*fn-name*⟩ ::= '`"`' [⟨*namespace*⟩'`::`'] [⟨*class*⟩'`::`'] ⟨*function*⟩ '`"`'
⟨*namespace*⟩, ⟨*class*⟩, ⟨*function*⟩ ::= *string*

Pre-defined Properties

Goanna provides the following pre-defined properties and their leagal values for specifying function parameters:

`ptr-read` (`true`, `maybe`, `false`)
Whether the value pointed to by the pointer parameter is read.

`ptr-written` (`true`, `maybe`, `false`)
Whether the memory location pointed to by the pointer parameter is written.

`ptr-freed` (`true`, `maybe`, `false`)
Whether the pointer parameter is freed.

`ptr-deleted` (`true`, `maybe`, `false`)
Whether the pointer parameter is freed by C++ `operator delete`.

`null-deref` (`maybe`, `false`)
Whether the pointer parameter is dereferenced without a null test.

`divisor` (`true`, `false`)
Whether the parameter is used as a divisor.

`strconst` (`true`, `false`)
Whether the memory location pointed to by the pointer parameter is written with a string constant.

`file-closed` (`true`, `false`)
Whether the file pointed to by the pointer parameter is closed.

`file-written` (`true`, `false`)
Whether the file pointed to by the pointer parameter is written.

`file-path` (`true`, `false`)
Whether the parameter is used as a path to a file.

`lib-path` (`true`, `false`)
Whether the parameter is used as a path to a dynamic library.

`relative-path` (`true`, `false`)
Whether the memory location pointed to by the pointer parameter is written with a Win32 relative path.

`sql-statement` (`true`, `false`)
Whether the parameter is used in an SQL query.

`os-command` (`true`, `false`)
Whether the parameter is used to run command on system.

`credential` (`true`, `false`)
Whether the parameter is used as a credential.

`format-str` (`true`, `false`)
Whether the parameter is used as a format string in functions such as `printf`.

`xpath-query` (`true`, `false`)
Whether the parameter is used as an XPath query.

`strcpy-source` (`true`, `false`)
Whether the parameter is used as string copy source.

`taint-source` (`true`, `false`)
Whether the memory location pointed to by the pointer parameter is written with a value tainted by user input.

`taint-overwritten` (`true`, `false`)
Whether the parameter's taint attribute is overwritten in the function.

`ldap-query` (`true`, `false`)
: Whether the parameter is used in an LDAP query.

`loop-cond-var` (`true`, `false`)
: Whether the parameter is used in a loop condition.

`memcpy-arg` (`true`, `false`)
: Whether the parameter is used as memory copy length.

`alloc-size` (`true`, `false`)
: Whether the parameter is used as a size in memory allocation.

`array-access` (`true`, `false`)
: Whether the parameter is used as an array index.

`buffer-offset` (`true`, `false`)
: Whether the parameter is used as a pointer offset.

`req-gt-z` (`true`, `false`)
: Whether the parameter p needs to be checked for `p > 0` before the call.

`req-ge-z` (`true`, `false`)
: Whether the parameter p needs to be checked for `p >= 0` before the call.

`req-ne-z` (`true`, `false`)
: Whether the parameter p needs to be checked for `p != 0` before the call.

`req-lt-1` (`true`, `false`)
: Whether the parameter p needs to be checked for `p < 1` before the call.

`req-le-1` (`true`, `false`)
: Whether the parameter p needs to be checked for `p <= 1` before the call.

`req-gt-n1` (`true`, `false`)
: Whether the parameter p needs to be checked for `p > -1` before the call.

`req-ge-n1` (`true`, `false`)
: Whether the parameter p needs to be checked for `p >= -1` before the call.

`req-le-ff` (`true`, `false`)
: Whether the parameter p needs to be checked for `p <= 255` before the call.

`req-gt-min` (`true`, `false`)
: Whether the parameter p needs to be checked for `p > INT_MIN` before the call.

The following two properties on function parameters are used to guide the analysis of the specified function itself, instead of affecting the analysis of the callers of the function. Thus they should not be used on API functions under the file name `"includes"`.

`value-interval` (*lower bound*, *upper bound*)
: The range of the parameter's possible input value represented by its lower and upper bounds in integers.

`array-size-interval` (*lower bound*, *upper bound*)
: When the parameter is an array pointer, the range of the input array size represented by its lower and upper bounds in integers.

Goanna provides the following pre-defined properties and their leagal values for specifying function side effects and return values:

`relative-path` (`true`, `false`)
: Whether the function returns a Win32 relative path.

`strconst` (`true`, `false`)
: Whether the function returns a string constant.

`taint-source` (`true`, `false`)
: Whether the function returns a value tainted by user input.

`alloc` (`maybe`, `false`)
: Whether the function allocates memory and returns a pointer to the allocated memory.

`null` (`maybe`, `false`)
: Whether the function returns `NULL`.

`chdir` (`true`, `false`)
: Whether the function calls `chdir()`.

`chroot` (`true`, `false`)
: Whether the function calls `chroot()`.

`impure` (`true`, `false`)
: Whether the function is impure.

`exit` (`true`, `maybe`, `false`)
: Whether the function terminates the calling process immediately.

`value-interval` (*lower bound*, *upper bound*)
: The range of the function's return value represented by its lower and upper bounds in integers.

`array-size-interval` (*lower bound*, *upper bound*)
: When returning an array pointer, the range of the array size represented by its lower and upper bounds in integers.

`return-ec` (`true`, `false`)
: Whether the function returns an error code.

`modify-errno` (`maybe`, `false`)
: Whether the function modifies `errno` defined in `errno.h`.

Return Codes

Here is a list of return codes `goanna-assume` returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goanna-assume` completed execution successfully.

- `goanna-assume` completes execution without errors
- `goanna-assume` prints help or version information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- The `--project-dir` option was given, but the `--goanna-dir` option was missing.
- `goanna-assume` encounters an error while parsing command-line arguments

11.9 goannacc and goannac++ – Analyze C/C++ Source Files

Synopsis

goannacc [*options*] ... [*compiler options*] ... [*file*] ...
goannac++ [*options*] ... [*compiler options*] ... [*file*] ...

Description

goannacc and goannac++ are the utilities that analyze individual C and C++ source files. In general, goannacc should be used for C source files, and goannac++ for C++ files.

goannacc and goannac++ can operate in two modes:

- **Analysis Only:** Perform analysis on the specified source files only.
- **Compile & Analyze:** Compile the specified source files, and then perform analysis.

For more details about these modes, see 10.2.

Command Line Options

--absolute-path Print absolute paths in warnings.

--all-checks Run all available checks (overrides all other check related options).

--brief-trace Show immediately relevant decisions in trace output, not the majority of decisions.

--c++ Indicate that file(s) contain C++ code.

--check=<*name*> Run a specific check (overrides any checks file).

--checks-file=<*file*> Use the checks listed in *<file>* instead of the default checks in `properties.init` file.

--checks=<*standard*> Run all checks in the specified coding standard. For example, `--checks=misrac2004` runs all available checks in the MISRA C:2004 standard.

--checks=<*standard*>-<*rule*> Run the check(s) corresponding to one rule in the specified coding standard. For example, `--checks=misrac2004-12.8` runs the check(s) that implement MISRA C:2004 rule 12.8.

--color, --colour Only available on Linux. Output in color.

--columns Print column positions in warnings.

--db=<*file*> Specify the database file to use for persistent information.

--directory=<*dir*> Before doing anything, change to *<dir>*.

--force-analysis Re-analyze files that have not changed since last run.

--help Print help message for common options.

--ignore-errors Ignore errors from the compiler.

--license-server=<*server[:port]*> Attempt to contact a license server at address *<server>*. *<Port>* is optional (defaults to 6200).

--no-compile, --nc Do not run the compiler.

--output-checks Output the checks that are currently loaded.

--output-format=<*format*> Specify a warning format used by Goanna to output warnings. The following special strings in *<format>* are expanded:

%FILENAME% the filename
%RELFILE% the filepath and filename
%RELPATH% the filepath
%ABSFILE% the absolute filepath and filename
%ABSPATH% the absolute filepath
%DBRELFILE% the filepath relative to the database file and filename

%DBRELPATH% the filepath relative to the database file
%LINENO% the line number
%COLUMN% the column number
%CHECKNAME% the check identifier
%SEVERITY% the checks severity rating
%MESSAGE% the warning message
%RULES% corresponding rule(s) from coding standards, if any
%TRACE% counter example if any
%FUNCTION% the function name
%SUPPRESSED% a * if the warning is suppressed
%WARNINGID% the hash of this warning
%EOL% a line break
%% a literal %.

The default warning format is: "`%RELFILE%:%LINENO%: warning: Goanna[%CHECKNAME%] Severity-%SEVERITY%, %MESSAGE%. %RULES%%EOL%`".

`--project-dir=<`*dir*`>` Only include header files in the given directory.

`--quiet` Only display warnings and no other output.

`--suppress=<`*warning id*`>` Suppress warning *<warning id>*.

`--suppression-status` Output suppression status markers, without suppressing warnings.

`--system-headers` Process system header files. Default: no.

`--timeout=<`*n*`>` Set a timeout (in seconds) for analysis of each source file. Default: 240.

Important: Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

`--trace` Prints out a trace through the function that leads to the warning. This is helpful for understanding why the warning occurs.

`--trace-format=<`*format*`>` Specify the format to output traces. The following special strings are used in the trace format:

%FILENAME% the filename
%RELFILE% the filepath and filename
%RELFILEX% the filepath and filename followed by "**:**", or blank if in the current source file
%RELPATH% the filepath
%ABSFILE% the absolute filepath and filename
%ABSFILEX% the absolute filepath and filename followed by "**:**", or blank if in the current source file
%ABSPATH% the absolute filepath
%DBRELFILE% the filepath relative to the database file and filename
%DBRELPATH% the filepath relative to the database file
%FUNCTION% the function name
%LINE% the line number
%TEXT% text describing the event on the trace
%TYPE% the type of the trace line
%EOL% a line break
%% a literal %.

The default trace format is: "`%LINE%: %TYPE% - %TEXT%%EOL%`".

`--unsuppress=<`*warning id*`>` Unsuppress warning *<warning id>*.

`--user-headers` Process user header files. Default: no.

`--verbose` Display additional output information.

`--version` Print version information.

`--warning-ids` Output warning hashes.

`--with-cc=<`*compiler*`>` Specify the C compiler executable to run (if `--nc` is not specified). Also affects the default dialect when no `--dialect` is specified.

`--with-cxx=<`*compiler*`>` Specify the C++ compiler executable to run (if `--nc` is not specified). Also affects the default dialect when no `--dialect` is specified.

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

`--32` Analyze code for 32-bit targets (longs and pointers are 32 bits wide).

`--64` Analyze code for 64-bit targets (longs and pointers are 64 bits wide).

`--advanced-help` Print help message for advanced options.

`--dialect=<`*file*`>` Specify the dialect of C/C++ compilers. Available dialects are:

```
armcc       metrowerks
c166        microsoft
c51         qnx
cc21k       renesas-h8
cosmic      renesas-rx
diab        ti-armcl
gnu         ti-cl16x
iar-8051    ti-cl2000
iar-arm     ti-cl430
iar-avr     ti-cl470
iar-msp430  ti-cl500
iar-m16c    ti-cl55
```

If you use this option, you should also specify `--with-cc`, `--with-cxx` and/or `--with-ld` to specify the paths to the compiler(s) and linker. If these are not specified, then Goanna will assume the default name for the specified dialect, which may not be what is available on your system.

If none of `--dialect`, `--with-cc`, `--with-cxx` or `--with-ld` are specified, then Goanna will assume the default of `gnu` dialect with `gcc` C compiler, `g++` C++ compiler and `ld` linker.

`--error` Exit with error status code when warnings emitted.

`--exclude=<`*file*`>` Exclude the specified *<file>* from analysis.

`--input-encoding=<`*type*`>` Specify the character encoding of the source file:

us-ascii ASCII (default)

utf-8 UTF-8

ansi (Available on Windows only) default character encoding of the system

`--internal-error=<`*value*`>` Exit with *<value>* on internal error.

`--ipa-iterations=<`*value*`>` Specify the number of times interprocedural analysis iterates towards a fixed point. The default is 2.

Important: Setting this value to 0 (meaning keep iterating until a fixed point is reached) is discouraged; this may cause Goanna to not terminate!

`--ipa-trace-depth=<`*value*`>` How many levels of inlining are performed for interprocedural traces. Default: 5.

Important: Setting this value to -1 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

`--issue-report=<`*type*`>` Control generation of issue report files:

never Never

on-failure On failures only

on-error On failures and analysis errors

timeout On failures, errors, and timeouts

always Always (even if successful)

`--license-borrow-hours=<`*number*`>` When contacting license server, borrow license for *<number>* of hours. Defaults to 1, maximum of 24.

`--license-dir=<`*directory*`>` Set directory in which to look for a license file.

`--no-globals` Do not analyze global integer variables.

`--no-ipa` Disable interprocedural analysis.

--output-file=<*file*> Append warning messages to a specified file.

--output-spec=<*file*> Use the contents of *<file>* as the output-format.

--parse-error-log=<*file*> Log parse errors to the specified file instead of stderr.

--timeout-error=<*value*> Exit with status code *<value>* when too many timeouts occur.

--timeout-limit=<*value*> Maximum number of per-phase timeouts. Default: 3.
Important: Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

--timeout-per-phase=<*n*> Set a timeout (in seconds) for each phase of analysis. This is useful if you have a few functions that take very long to analyze and you would like to limit the time spent on these, while still getting as many results as possible on everything else. Default: 60.
Important: Setting this value to 0 (meaning infinite) is discouraged; this may cause Goanna to not terminate!

Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

--alias

--configure=<*dialect*>

--dialect-mod=<*dialect-mod*>

--dataflow

--diagnostics-mode

--no-alias

--no-dataflow

--no-default-packages

--package=<*package*>

--package-dir=<*directory*>

Any unrecognized options will be passed through to the compiler as compiler arguments, unless `--no-compile` is specified, in which case they will be ignored.

Return Codes

Here is a list of return codes goannacc returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and goannacc completed execution successfully.

- goannacc completes execution without errors
- goannacc configures the default dialect
- goannacc prints help or version information
- goannacc invokes the C/C++ compiler, but is configured to ignore the compiler return code. goannacc returns `0` even if the compiler returns an error
- goannacc finds code warnings from source code analysis, and is not configured to return an error in the presense of code warnings (this is the default setting, also see return code 1 (One)_)

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- goannacc attempts to create or change into a directory that does not exist, or a permission error occurred
- goannacc performs an operation that requires the project database, but the database is not provided or not found
- An invalid license server is provided to goannacc
- An argument to goannacc needs to be a positive integer, but the user-provided value is not a positive integer. For example, the number of IPA[2] iterations, or timeout values

[2] Inter-Procedural Analysis

- goannacc encountered an error while parsing command-line arguments or the dialect file
- goannacc finds code warnings from source code analysis, and is configured to return an error in the presense of code warnings

127

- goannacc encounters an error when trying to invoke the compiler (for example, if the compiler cannot be found)

-1

- goannacc attempts to call goannald, but goannald does not exist or cannot be found

Inherited Return Values

- If goannacc invokes the C/C++ compiler, it returns the compilers's return code unless explicitly configured otherwise
- If goannacc invokes goannald, it returns goannald's return code unless explicitly configured otherwise

Configurable Return Values The following error modes have configurable return values.

- Internal Error: Goanna encounters an error internally and cannot complete analysis.
- Parser Error: The Goanna C/C++ parser cannot parse the input source code file. This is often attributed to syntax errors in the source code, or uncommon C/C++ language constructs that the Goanna parser does not recognise
- License Expired Error: The Goanna License has expired
- Timeout Error: Goanna is unable to complete before the timeout expired

11.10 goannald – C/C++ Link Time Analysis

Synopsis

goannald [*options*] ... [*linker options*] ... [*file*] ...

Description

goannald is the utility that performs link-time analysis on a whole program consisting of multiple source and/or object files, that have already been individually analyzed with goannacc and goannac++.

goannald can operate in two modes:

- **Analysis Only:** Perform link time analysis only.
- **Link & Analyze:** Link the specified object files, and then perform link time analysis.

For more details about these modes, see 10.2.

Command Line Options

--all-checks Run all available checks (overrides all other check related options).

--check=<*name*> Run a specific check (overrides any checks file).

--checks-file=<*file*> Use the checks listed in *<file>* instead of the default checks in properties.init file.

--checks=<*standard*> Run all checks in the specified coding standard. For example, --checks=misrac2004 runs all available checks in the MISRA C:2004 standard.

--checks=<*standard*>-<*rule*> Run the check(s) corresponding to one rule in the specified coding standard. For example, --checks=misrac2004-12.8 runs the check(s) that implement MISRA C:2004 rule 12.8.

--color, --colour Only available on Linux. Output in color.

--columns Print column positions in warnings.

--db=<*file*> Specify the database file to use for persistent information.

--help Print help message for common options.

--no-compile, --nc Do not run the compiler.

--verbose Display additional output information.

--version Print version information.

--warning-ids Output warning hashes.

--with-ld=<*linker*> Specify the C/C++ linker executable to run (if --nc is not specified).

Advanced Command Line Options

The following options are intended to be used only in cases where your environment requires them. In general, you do not need to use these options.

--dialect=<*file*> Specify the dialect of C/C++ compilers. Available dialects are:

```
armcc       metrowerks
c166        microsoft
c51         qnx
cc21k       renesas-h8
cosmic      renesas-rx
diab        ti-armcl
gnu         ti-cl16x
iar-8051    ti-cl2000
iar-arm     ti-cl430
iar-avr     ti-cl470
iar-msp430  ti-cl500
iar-m16c    ti-cl55
```

If you use this option, you should also specify `--with-cc`, `--with-cxx` and/or `--with-ld` to specify the paths to the compiler(s) and linker. If these are not specified, then Goanna will assume the default name for the specified dialect, which may not be what is available on your system.

If none of `--dialect`, `--with-cc`, `--with-cxx` or `--with-ld` are specified, then Goanna will assume the default of `gnu` dialect with `gcc` C compiler, `g++` C++ compiler and `ld` linker.

`--error` Exit with error status code when warnings emitted.

`--exclude=<`*file*`>` Exclude the specified *<file>* from analysis.

Diagnostics Command Line Options

The following options are provided for diagnostics purposes only. **Do not use these options unless directed by Red Lizard Software support team.**

`--diagnostics-mode`

Any unrecognized options will be passed through to the linker as linker arguments, unless `--nc` is specified, in which case they will be ignored.

Return Codes

Here is a list of return codes `goannald` returns, listed as headings. Under the headings are a list of modes that provides that return code.

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goannald` completed execution successfully.

- `goannald` completes execution without errors
- `goannald` prints help or version information
- `goannald` invokes the C/C++ linker, but is configured to ignore the linker return code. `goannald` returns `0` even if the linker returns an error
- `goannald` finds code warnings from source code analysis, and is not configured to return an error in the presense of code warnings (this is the default setting, also see return code 1 (One)_)

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- `goannald` encountered an error while parsing command-line arguments
- `goannald` performs an operation that requires the project database, but the database is not provided or not found
- `goannald` finds code warnings from source code analysis, and is configured to return an error in the presense of code warnings

Inherited Return Values

- If `goannald` invokes the C/C++ linker, it returns the linker's return code unless explicitly configured otherwise

Configured Return Values The following error modes have configurable return values.

- Internal Error: Goanna encounters an error internally and cannot complete analysis.

11.11 `goreporter` – Goanna Dashboard Server and Administration Tool

Synopsis

`goreporter start-server` [*options*]
`goreporter stop-server` [*options*]
`goreporter db-upgrade` [*options*]

Description

`goreporter` is the command line utility used to drive the Goanna Dashboard. It has a number of commands:

`start-server` Starts the Goanna Dashboard server

`stop-server` Stops the Goanna Dashboard server (when started with the `start-server`) command.

`db-upgrade` Perform the database upgrade; see 9.5 for details about this feature.

Running The Goanna Dashboard

The Goanna Dashboard is accessible through the embedded web server `goreporter`. The server can be run in two ways; either as a standalone binary being executed from the command line, or as a Linux or Windows Service.

Standalone Use

To start the server run the following command:

```
goreporter start-server
```

When `goreporter` starts the server, it will display the port the server is running on. The port can be specified by using the `--port` flag. By default the server starts at 1197 and increments until it finds a free port. It is only possible to run one server instance at a time using this command.

To access the server, open a web browser and browse to http://localhost:<port>, where `<port>` is the port number `goreporter` starts the server on. To stop the server, run the command:

```
goreporter stop-server
```

Linux Service

Note: Using Linux service feature requires Goanna Central to be installed under root (often with `sudo` command).

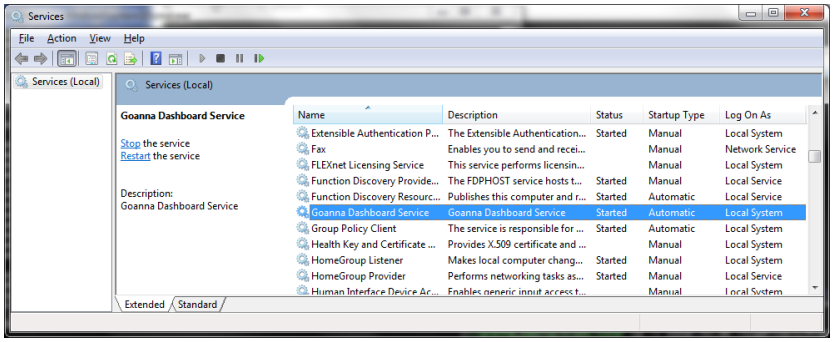
A service script for the web server is installed as part of the Goanna installation process (if installed with `sudo` or from within the root terminal). It uses the configuration file located at `/etc/goreporter.conf`. This file is used to configure the server, and takes the same options that the `goreporter` executable does.

The service is installed in either `/etc/init.d` or `/etc/rc.d`, depending on your distribution. To start the service on a distribution using `init.d` use:

```
/etc/init.d/goreporterd {start|stop|reset|restart|force-restart}
```

Windows Service

The Windows service is installed and started on installation of Goanna Central. A configuration file is stored in the same location as your summary database. To change settings with the service, modify the configuration file and then restart the service through the Windows Service Manager.


Services
File Action View Help
Services (Local)
Goanna Dashboard Service
Stop the service
Restart the service
Description:
Goanna Dashboard Service
Name
Description
Status
Startup Type
Log On As
Extensible Authentication P... The Extensible Authentication... Started Manual Local System
Fax Enables you to send and recei... Manual Network Service
FLEXnet Licensing Service This service performs licensin... Manual Local System
Function Discovery Provide... The FDPHOST service hosts t... Started Manual Local Service
Function Discovery Resourc... Publishes this computer and r... Started Automatic Local Service
Goanna Dashboard Service Goanna Dashboard Service Started Automatic Local System
Group Policy Client The service is responsible for ... Started Automatic Local System
Health Key and Certificate ... Provides X.509 certificate and ... Manual Local System
HomeGroup Listener Makes local computer chang... Started Manual Local System
HomeGroup Provider Performs networking tasks as... Started Manual Local Service
Extended
Standard

Configuration File

To point `goreporter` to a configuration file, use the following command:

```
goreporter start-server --config=goreporter.conf &
```

Where `goreporter.conf` is the name of your configuration file.

`goreporter` configuration files support all the options specified on the command line. The most commonly used of these options are:

`summary-db` the location of the summary database

`port` the port to run the server on

`log` the location to put the goreporter log file

They are specified as `<option>=<value>` pairs, with `#` used for comments.

Running the Dashboard as a Standalone Web Server

The Goanna Dashboard server can be run as a web server for local intranet or remote access in two ways. If you want to run the server on a machine with no other web server running on it, the simplest way to set up a GoReporter server is to run the server either as a service or as a standalone instance with the port set to 80 (standard web port).

Provided no firewall or similar is blocking access, the Goanna Dashboard will now be visible by browsing to http://<yourserveraddress>. If you run the server standalone and the computer you're running the server on is restarted, the Goanna Dashboard server will not automatically be reset.

Running the Dashboard through Apache

It is also possible to use the Goanna Dashboard server in conjunction with your existing web server. The following example shows how to do this with Apache web server.

The way to run the Goanna Dashboard with existing Apache installation is to run `goreporter` on the different port than Apache (Apache normally uses port 80), and then use Apache's `mod_proxy` to forward all traffic on the special Dashboard URL to `goreporter`.

Firstly, modify your `httpd.conf` to allow proxy access. For example, if your site is located at http://mysite.com and you want to navigate to the Dashboard through the url http://mysite.com/reporter, then add these lines into `httpd.conf`:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
...
ProxyRequests Off
ProxyPass /reporter http://localhost:1197
ProxyPassReverse /reporter http://localhost:1197
```

Then restart Apache.

You can then start the Goanna Dashboard server either as a service or as a standalone instance using the port 1197.

It is important to specify the port number when running `goreporter` in this instance, as Apache has been configured to only connect to `goreporter` on this port.

The Goanna Dashboard is now available at http://mysite.com/reporter/.

Goanna Dashboard Database (Summary Database)

Goanna Dashboard uses its own database (called *summary database*, or *summary DB*) to store details about your project. By default this database is stored in:

- `<installed-directory>/reporter` on Linux where the Goanna installation directory is `<installed-directory>`, or
- `C:\Documents and Settings\<username>\Local Settings\Application Data\RedLizards\Goanna Central\` for Windows XP and Server 2003, or
- `C:\Users\<username>\AppData\Local\RedLizards\Goanna Central\` for Windows Vista, Server 2008 and later, where `<username>` is the name of the user account who installed Goanna Central.

It is possible to change where the summary database is stored by using the `--summary-db=<path>` flag. However, if you do change the default summary database location, be sure to specify new location of the database with either:

- through `--summary-db` flag when you run Goanna commands, or
- by specifying the new location in the `goreporter` configuration file.

Integrating Reporting into Your Build

To capture an anlysis snapshot, use the tool `goanna-snapshot`.

Important: Once the snapshot is taken, make sure that the project database stays at the same location. This is because some features of the Goanna Dashboard relies on the project database being always available at the same location.

Command Line Arguments

`start-server`

`--summary-db=<`*file*`>` The database in which to save snapshots.

`--port=<`*port number*`>` Port number (default 1197, if 1197 is not available then 1198, 1199...). If this option is set and the specified port is unavailable then the server will quit.

`--log=<`*true/false*`>` Port number (default 1197, if 1197 is not available then 1198, 1199...). Toggles logging to stdout. Default true

`--log-file=<`*path to log file*`>` Toggles logging to a file.

`--config=<`*path to configuration file*`>` When specified overwrites the given options with those specified in the file

`db-upgrade`

Note: See 9.5 for details about this feature.

`--summary-db=<`*file*`>` The database in which to save snapshots.

`--progress=<path to progress>` Specify the file to write progress information into.

`--feature=<feature>` Specify the feature(s) to upgrade; valid values are `auto_vacuum` and `warning_index`.

Return Codes

0 (Zero) A return code of `0` generally means that no errors were encountered, and `goreporter` completed execution successfully.

- `goreporter` completes execution without errors
- `goreporter` prints help or usage information

1 (One) A return code of `1` generally means that a user-provided parameter is invalid.

- `goreporter` encounters an error while parsing command-line arguments
- `goreporter` attempts to read or write to a file, but the file cannot be found or `goreporter` does not have the correct permissons
- `goreporter` is given a database argument, but the database is not a valid Goanna project database
- `goreporter` attempts to start the web server process on a particular TCP/IP port, but the port is already in use

2

- `goreporter` encounters an unexpected or internal error when processing a subcommand

Index

_GOANNA preprocessor symbol, 34
goanna-assume, 58
assert macro, 34
goanna-analyse, 50
goanna-init, 48
goanna-metrics, 56
goanna-package, 55
goanna-report, 53
goanna-snapshot, 54
goanna-trace, 49
goannac++, 64
goannacc, 64
goannald, 69
goreporter, 71
GoannaKnowledge Base File Format, 58
GoannaKnowledge Base Properties, 61

Checks
  Changing check set, 25
  Custom Packages, 25
  Disabling Packages, 25
  Enabling Packages, 24
  Listing Packages, 24
  Packages, 24
  Setting, 25
Command Line Options
  --, 49, 51, 56
  --32, 66
  --64, 66
  --absolute-path, 64
  --advanced-help, 48, 49, 52–54, 56, 66
  --alias, 67
  --all-checks, 50, 64, 69
  --all-metrics, 56
  --api-list, 58
  --as-api, 58
  --brief-trace, 64
  --c++, 64
  --check, 50, 64, 69
  --checks, 50, 64, 69
  --checks-file, 64, 69
  --color, --colour, 64, 69
  --columns, 64, 69
  --config, 73
  --configure, 67
  --dataflow, 67
  --db, 64, 69
  --delimiter, 53
  --diagnostics-mode, 67, 70
  --dialect, 65, 66, 69
  --dialect-mod, 67
  --directory, 64
  --dryrun, 50
  --error, 66, 70
  --exclude, 50, 51, 56, 66, 70
  --feature, 74
  --force-analysis, 50, 64
  --force-tracing, 49
  --goanna-dir, 48–50, 52–58, 63
  --header-protos, 50
  --help, 48, 49, 51, 53, 54, 56, 64, 69
  --ignore-errors, 64
  --input-encoding, 66
  --internal-error, 66
  --ipa-iterations, 33, 66
  --ipa-trace-depth, 66
  --issue-report, 52, 66
  --jobs, 50, 56
  --latest, 53
  --license-borrow-hours, 49, 50, 56, 66
  --license-dir, 49, 51, 56, 66
  --license-server, 49, 50, 56, 64
  --log, 73
  --log-file, 73
  --match, 50, 51, 56
  --metric, 56
  --nc, 65, 69, 70
  --no-alias, 67
  --no-compile, 67
  --no-compile, --nc, 64, 69
  --no-dataflow, 67
  --no-default-packages, 67
  --no-globals, 66
  --no-header-protos, 50
  --no-ipa, 33, 66
  --no-system-headers, 50
  --no-user-headers, 50
  --output-checks, 64
  --output-file, 67
  --output-format, 50, 64
  --output-spec, 67
  --package, 67
  --package-dir, 67
  --parse-error-log, 67
  --port, 73
  --progress, 74
  --project-dir, 48–50, 52–58, 63, 65
  --project-name, 48
  --quiet, 65
  --rel-path, 58
  --report-group, 53
  --summary-db, 73
  --suppress, 65
  --suppression-status, 65
  --system-headers, 50, 65
  --timeout, 50, 65
  --timeout-error, 67

--timeout-limit, 67
--timeout-per-phase, 52, 67
--trace, 51, 65
--trace-format, 51, 65
--ungroup, 50, 56
--unsuppress, 65
--user-headers, 50, 65
--verbose, 65, 69
--version, 48, 49, 51, 53, 54, 56, 65, 69
--warning-ids, 65, 69
--with-cc, 65
--with-cxx, 65
--with-ld, 69
Common Weakness Enumeration (CWE), 26
Computer Emergency Response Team (CERT) C/C++ Coding Standard, 26

Database
Project, 46
Summary, 73
Dialect, 47

False positives, 33
Bug status, 37

Goanna Dashboard, 37
Bug status, 37
Server, 71
Snapshot generation, 73
Views, 37

Installation
Linux, 12
Windows, 13
Interprocedural analysis, 33

License, 12
Activation, 13

Motor Industry Software Reliability Association (MISRA) C++:2008, 26
Motor Industry Software Reliability Association (MISRA) C:2004, 26

Running analysis from IAR Embedded Workbench, 29
Running analysis from Keil μVision, 31

Sample Code, 34
Standards, 26

Traces
Dashboard, 41

Warning
Suppression, 37, 40